G. Weck

Der Text-Editor SB&EDIT

Beschreibung der Version (4.08)

(Sprachversion 2)

Fachbereich
Angewandte Mathematik
und Informatik
der Universität des Saarlandes

D-6600 Saarbrücken

September 1978

Bericht Nr. A78-15

Inhalt
======

6.10.78
We

Vorwort

Der vorliegende Bericht beschreibt die Implementierung eines integrierten Texthaltungs-Systems auf der Basis des MIT-Editors TECO. Dieser Editor wurde den im Rechenzentrum der Universitaet des Saarlandes gegebenen Umgebungsbedingungen angepasst und zunaechst fuer den Rechner TR440 implementiert. Die Implementierung wurde dabei so vorgenommen, dass sie ohne grosse Schwierigkeiten auf andere Rechner uebertragbar ist, insbesondere auch in einen Betrieb mit abgesetzten intelligenten Arbeitsplaetzen eingebaut werden kann. An dieser Stelle ist noch der Hinweis angebracht, dass zur Implementierung des Editors ausser einer Liste der TECO-Kommandos (Anhang C der TECO-Beschreibung) keine weiteren Unterlagen ueber TECO zur Implementierung benoetigt und verwendet wurden; die Implementierung konnte vollstaendig frei unter Benutzung allein der Kenntnisse ueber den Umgang mit TECO erfolgen.

Der Bericht ist so aufgebaut, dass er einerseits die zugrundeliegenden Konzepte dieses Editor-Systems verdeutlicht, andererseits aber auch als Nachschlagwerk fuer den Benutzer des Editors verwendet werden kann. Zur Eingabe und Aufbereitung des Textes dieses Berichtes wurde der im Bericht beschriebene Editor selbst verwendet.

Es ist mir ein Vergnuegen, allen denen zu danken, die mich bei der Implementierung des Editors unterstuetzt haben und geholfen haben, so manche dabei auftretende Frage zu klaeren. Besonderen Dank verdient hier Herr B. Kett, ohne dessen Beistand es mit Sicherheit nicht moeglich gewesen waere, dieses Programm in der relativ kurzen Zeit von einem halben Jahr fertigzustellen. Ebenso moechte ich Herrn L. Gerlach dafuer danken, dass er mir freundlicherweise ein Programm zur formatierten Ausgabe von Texten zur Verfuegung gestellt hat, mit dessen Hilfe dieser Bericht gedruckt wurde.

6.10.78
We

Literatur
=========

[1] dec-system-10: Getting Started with TOPS-10 Commands, Digital Equipment Corporation, Maynard (Mass.), Schrift DEC-10-OTSCA-A-D, Juni 1975

[2] dec-system-10: Introduction to TECO (Text Editor and Corrector, Digital Equipment Corporation, Maynard (Mass.), Schrift DEC-10-UTECA-A-D, Juni 1975

[3] Jacob Palme: TOPSTEACH, A computer-aided introductory course to the DEC 10 system, Research Institute of National Defense, Stockholm, Oktober 1973

[4] dec-system-10: Users Handbook, Digital Equipment Corporation, Maynard (Mass.)

0. Einleitung

0.1. Ein alternatives Texthaltungs-System

Texte - z.B. Quell-Programme - werden auf der TR440 ueblicherweise in sogenannten Texthaltungsdateien (meist RAM-Dateien mit Satzbau U800 oder aehnlich) gehalten. Zu ihrer Verarbeitung stellt das Programmiersystem mit dem Operator PS&TEXTHALT eine Reihe von Dienstleistungen zur Verfuegung, zu denen Eintragen, Kopieren, Korrigieren und Loeschen von Saetzen (Zeilen) gehoeren. Diese Form der Texthaltung hat vier gravierende Nachteile:

- Sie ist vom Speicherplatz auf dem Hintergrund her sehr aufwendig, da fuer jeden Text, egal wie kurz er ist, eine eigene Datei erzeugt werden muss. Diese Datei muss in einen Katalog eingetragen werden, und als RAM-Datei muss sie ein Stellvertretergebiet enthalten, so dass der minimale Platzbedarf immerhin 2 bis 3 K Ganzworte betraegt, waehrend die eigentliche Nutzinformation, der Text, oft nur einige hundert Ganzworte umfasst. Dies bedeutet insbesondere bei der LFD eine nicht unbetraechtliche Platzverschwendung.

- Die von der Texthaltung zur Verfuegung gestellten Operationen sind satzorientiert. Es ist daher nicht moeglich, Saetze aufzubrechen oder zu konkatenieren, was oft gerade bei Programm-Aenderungen erwuenscht ist. Ausserdem sind beim Korrigieren von Texten Aenderungen lokal begrenzt; wenn eine Aenderung in aehnlicher Form an mehreren Stellen auszufuehren ist, so muessen diese Stellen jeweils einzeln lokalisiert und geaendert werden, was eine recht zeitraubende und stumpfsinnige Arbeit werden kann.

- Die Numerierung der Saetze der Datei, also der Textzeilen, macht es unmoeglich, mehr als eine bestimmte Anzahl von Zeilen zwischen zwei existierende Zeilen einzuschieben, ohne die ganze Datei neu durchzunumerieren. Der Zugriff auf die Saetze der Datei erfolgt ueber diese Satznummern, so dass der Benutzer der Texthaltung auf ein Listing seines Textes mit gueltigen Satznummern angewiesen ist - andererseits ist er aber beim Eintragen groesserer Textstuecke aus dem genannten Grund oft gezwungen, eben diese Satznummern zu veraendern.

- Zur Bearbeitung einer Texthaltungsdatei stehen dem Benutzer eine Vielzahl von Programmiersystem-Kommandos zur Verfuegung, so dass er gezwungen ist, fuer aehnliche Operationen verschiedene Kommandos zu geben und damit jedesmal einen Entschluesslerlauf und einen Operatorstart zu veranlassen. Es waere einfacher und uebersichtlicher, wenn der Operator PS&TEXTHALT zu Beginn der Textverarbeitung mit-

tels eines einzigen Kommandos gestartet wuerde und dann die Spezifikation der Operationen mittels Dialog-Eingabe erlaubte.

Im vorliegenden Bericht soll nun ein Texthaltungssystem beschrieben werden, das die genannten Nachteile vermeidet. Die beiden Komponenten dieses Systems sind:

- das Komprimierprogramm SB&COSY,
- der Text-Editor SB&EDIT.

Mit diesen beiden Programmen ist es moeglich, eine eigene, raumsparende und leistungsfaehige Texthaltung aufzubauen. Dieser Bericht beschreibt hauptsaechlich den Text-Editor SB&EDIT, den der Benutzer zur Bearbeitung seiner Texte einzusetzen hat. Das Dienstprogramm SB&COSY, das in diesem Zusammenhang nur zur Kompression bestehender Texthaltungsdateien gebraucht wird, wird hier lediglich kurz beschrieben, zusammen mit der Struktur der erzeugten Dateien.

Dieses alternative Texthaltungssystem hat gegenueber der satzorientierten konventionellen Texthaltung noch zusaetzlich den Vorteil, dass es sich leicht so erweitern laesst, dass der Text-Editor off-line in einem intelligenten (mikroprozessorgesteuerten) Arbeitsplatz laeuft, der nur gelegentlich mit dem Rechner oder seiner eigenen Peripherie Daten auszutauschen braucht. Die dazu noetigen Erweiterungen des Editors werden am Schluss dieses Berichts beschrieben.

20. 9.78
We

0.2. Die Komponenten des Texthaltungs-Systems

0.2.1. Texthaltung in COSY-Deck-Dateien

Um Platz auf dem Hintergrund - insbesondere auf der LFD - zu sparen, empfiehlt es sich, groessere Mengen von Texten - z.B. Quell-Programme - nicht in Texthaltungsdateien nach TR440-Standard (also RAM-Dateien mit einem Satzbau U800 oder aehnlich) zu speichern, sondern die Texte in komprimierter Form in PHYS-Dateien mit minimalem Verschnitt zu speichern. Man benoetigt dazu ein Programm, das es ermoeglicht, Texthaltungsdateien in die komprimierte Form ueberzufuehren und umgekehrt komprimierte Dateien in Texthaltungsdateien zu entzerren.

Ein solches Komprimierungs-Programm wird in Saarbruecken mit Erfolg eingesetzt; es handelt sich dabei um den Operator SB&COSY, der eine eigene Datenstruktur in PHYS- (und auch SEQ-) Dateien unterstuetzt, die es (im Falle von PHYS-Dateien) gestattet, bis zu 49 verschiedene Texte separat in einer Datei zu halten. Mit dem Operator SB&COSY koennen Texthaltungsdateien komprimiert und entzerrt werden; man kann sich ueber die Menge der Textstuecke (genannt COSY-Decks) in der PHYS-Datei informieren, und man kann Decks aus der PHYS-Datei loeschen. Dazu steht das im Abschnitt 0.3.2. beschriebene Kommando #COSY zur Verfuegung.

Die PHYS-Datei, die die komprimierten Texte enthaelt, hat den folgenden Aufbau:

- Block 0 der Datei ist ein Steuerblock, der Informationen zur Verwaltung der Decks in der Datei enthaelt:
  - die Namen der abgespeicherten Decks; als Namen sind beliebige Standardnamen mit bis zu 12 Zeichen erlaubt;
  - zu jedem Deck die Nummer des ersten Blocks der Datei, der zu diesem Deck gehoert;
  - zu jedem Deck der Komprimierungs-Modus, in dem das Deck erzeugt wurde (K6, K8 oder Z8 - siehe Abschnitt 0.3.2.);
  - die Anzahl der Decks in der Datei;
  - die Nummer des ersten freien Blocks in der Datei; hier beginnt ein neu erzeugtes Deck.

- Die restlichen Bloecke der Datei enthalten die Decks in lueckenloser Reihenfolge. Dabei gilt fuer die Form, in der die einzelnen Decks abgespeichert sind, das Folgende:
  - Jedes Deck beginnt an einem Block-Anfang und erstreckt sich ueber die folgenden Bloecke; dabei sind alle Bloekke mit eventueller Ausnahme des letzten ganz mit Text gefuellt.
  - Das erste Wort jedes Blocks enthaelt Verwaltungsinformation, und zwar die laufende Nummer des Blocks im Deck, eine sogenannte Kartenkennung, die angibt, ob es der letzte Block im Deck ist, und eine Pruefsumme, die einen

gewissen Schutz gegen Fehluebertragungen darstellt. Der Text selbst enthaelt keinerlei sonstige Verwaltungsinformation, insbesondere auch keine Satzstruktur.
- Folgen von Zwischenraeumen (Blanks) im Text sind durch ein Sonderzeichen und die Anzahl der Blanks, also nur durch zwei Zeichen repraesentiert; Zeilenwechsel ebenso nur durch zwei Sonderzeichen. Diese Kompression ist voll transparent; es gibt also keine Folgen von ZC1-Zeichen, die nicht so komprimiert wuerden, dass sie nicht wieder in die Original-Folge entzerrt werden koennen - dies gilt auch fuer Steuerzeichen, die in ueblichen Texten nicht vorkommen.

Bei der Kompression eines Textes kann man einen Kompressionsmodus angeben, der bestimmt, ob man sich auf einen eingeschraenkten Zeichenvorrat bezieht (Modus K6) oder ob man alle Zeichen des Zentralcodes ZC1 zur Verfuegung haben will (Modus K8). Bei vollem Zeichenvorrat kann man schliesslich noch veranlassen, dass die Zeilennummern der Texthaltungs-Datei mit in den komprimierten Text uebernommen werden (Modus Z8).

20. 9.78
We

0.2.2. Textverarbeitung in COSY-Deck-Dateien

Fuer die Verarbeitung von Texten, die in der COSY-Deck-Struktur abgespeichert sind, steht ein Text-Editor zur Verfuegung, dessen Bedienung im vorliegenden Bericht beschrieben wird. Dieser Editor SB&EDIT verarbeitet direkt die Decks einer COSY-PHYS-Datei; zur Textverarbeitung ist es also nicht noetig, die Decks zuerst in eine Texthaltungsdatei nach TR440-Konvention zu entzerren und dann diese zu bearbeiten.

Die vom Editor SB&EDIT zur Verfuegung gestellten Dienstleistungen umfassen die zur Textverarbeitung benoetigten Operationen Eintragen, Kopieren, Korrigieren und Loeschen, dies aber im Gegensatz zur TR440-Texthaltung nicht auf der Basis einzelner Saetze (Zeilen) oder Mengen von Saetzen, sondern kontext-orientiert auf der Basis beliebiger Strings, die sich ueber beliebig viele Zeichen und/oder Zeilen erstrecken koennen. Die gewuenschten Textverarbeitungsfunktionen werden mittels einer eigenen Kommandosprache dem Editor spezifiziert. Ein Textverarbeitungs-Lauf kann dabei beliebig viele dieser Funktionen anstossen und entsprechend viele dieser Editor-Kommandos entgegennehmen.

Der Editor wird durch das im naechsten Abschnitt beschriebene Kommando #EDIT gestartet; dabei wird im Parameter DATEI dieses Kommandos die zu bearbeitende COSY-Datei spezifiziert. Diese Datei kann eine beliebige COSY-Deck-Datei vom Typ PHYS, G128W sein; der Editor bearbeitet jedoch nur im Modus K8 oder Z8 in diese Datei komprimierte Decks. LF- oder WSP- (speziell POOL-) Dateien muessen zur Bearbeitung angemeldet sein, wenn man das Kommando #EDIT gibt. Ist die Datei nur zum Lesen angemeldet, so erlaubt der Editor entsprechend auch nur lesende Zugriffe auf die Datei. Ist die Datei leer und ist Schreibzugriff erlaubt, so wird eine (zunaechst leere) COSY-Struktur in ihr erzeugt.

Die beiden naechsten Abschnitte beschreiben die benoetigten Kommandos #EDIT und #COSY. Im Kapitel 1 werden dann die Entwurfskriterien fuer den Aufbau des Editors und die der Editor-Kommandosprache zugrundeliegenden Konzepte beschrieben; dieses Kapitel ist zum Verstaendnis von grundlegender Bedeutung. Das 2. Kapitel definiert dann die Bestandteile der Kommandosprache in einer Form, die als Nachschlagwerk zur Textverarbeitung mit dem Editor gedacht ist. Eine Liste aller Fehlermeldungen des Editors sowie einige Beispiele haeufig benoetigter Operationen in der Editor-Sprache ergaenzen diese Definitionen. Das Kapitel 3 macht schliesslich noch einige Aussagen zur vorliegenden Implementierung und zur Uebertragung dieser Implementierung auf intelligente Arbeitsplaetze.

0.3. Kommandos des Programmiersystems

0.3.1. Das Kommando #EDIT

Editieren von COSY-Deck-Dateien EDIT

DATEI= Name der zu editierenden Datei
- datei: Datei in der Standarddatenbasis
- db.datei: Datei in der Datenbasis db

DECK= zu editierendes Deck
- \- : * Voreinstellung: Zu editierende Decks muessen explizit durch Editor-Kommandos zur Bearbeitung eroeffnet werden
- deck: Falls deck nicht existiert, wird ein leeres Deck mit diesem Namen zum Schreiben eroeffnet, andernfalls wird das Deck deck zur Bearbeitung im Back-up-Betrieb eroeffnet und die erste Seite eingelesen

AUSGABE= Ausgabe-Datei fuer COSY-Entzerren durch die Kommandos EL und EP
- \- : Es wird nicht in eine Texthaltungsdatei entzerrt
- -STD- : * Voreinstellung: Die Texthaltungsdatei fuer das Entzerren liegt in der Standarddatenbasis bzw. wird dort erzeugt und hat den Namen des bearbeiteten Decks
- datei: Die Texthaltungsdatei liegt in der Standarddatenbasis bzw. wird dort erzeugt und hat den Namen datei
- db.datei: Die Texthaltungsdatei liegt in der Datenbasis db bzw. wird dort erzeugt und hat den Namen datei

PROTOKOLL= Angaben zur Protokollierung beim Entzerren mit den Kommandos EL und EP
- \- : * Voreinstellung: kein Protokoll
- -STD- : Standard-Protokoll
- KO: zusaetzliches Protokoll auf dem Terminal

0.3.2. Das Kommando #COSY

Entzerren und Komprimieren von Oktaden-Dateien — COSY --

QUELLE=		Datei, die die zu verarbeitende Information enthaelt
	datei	Datei in der Standarddatenbasis
	kat.datei	Datei aus dem Katalog kat
	datei-p	Datei mit Passwort p in der Standarddatenbasis
	kat.datei-p	Datei mit Passwort p aus dem Katalog kat
ZIEL=		Datei, die die umgewandelte Information aufnimmt
	datei	Datei aus der Standarddatenbasis
	kat.datei	Datei aus dem Katalog kat
	datei-p	Datei mit Passwort p in der Standarddatenbasis
	kat.datei-p	Datei mit Passwort p aus dem Katalog kat
		Die COSY-Datei muss vom Typ PHYS, G128W sein (Wenn man auf die Deck-Struktur verzichtet, ist auch SEQ, G20W moeglich; der Editor verlangt jedoch PHYS-Dateien.); die Texthaltungsdatei muss vom Typ RAM, Un0 (n z.B. = 80) oder SEQ, Un0 sein. Existiert die bei ZIEL angegebene Datei beim Entzerren nicht, so wird eine Datei gleichen Namens kreiert. Die COSY-Datei wird gegebenenfalls selbsttaetig zur Bearbeitung angemeldet und dann nach dem COSY-Lauf wieder abgemeldet.
MODUS=		Art des Entzerrens bzw. Komprimierens
	E *	Voreinstellung: Entzerren in dem Modus, in dem das betreffende Deck komprimiert wurde
	E6	Entzerren von 6 Bit/Zeichen
	E8	Entzerren von Oktaden
	K6	Komprimieren auf 6 Bit/Zeichen (kleiner Zeichenvorrat)
	K8	Komprimieren auf 8 Bit/Zeichen (voller Zeichenvorrat des ZC1)
	Z8	Komprimieren auf 8 Bit/Zeichen unter Erhaltung der Zeilennummer
	L	Loeschen eines Decks. Dabei muss

```
                die das Deck enthaltende Datei un-
                ter ZIEL angegeben sein.
  I             Informieren ueber die Decks in der
                COSY-Datei; diese muss unter QUELLE
                angegeben sein.

-------------------------------------------------

KORREKTUREN=    Angaben zur Modifizierung der
                Quelldatei
  datei         Die Datei datei der Datenbasis db
  db.datei      enthaelt die Steuerkarten zum Mo-
                difizieren der Quelldatei

DECKNAME=       Angabe von Decks in PHYS-Dateien
  -           * Voreinstellung: Der Deckname ist
                gleich dem Namen der Texthaltungs-
                datei.
  name          Es soll aus dem Deck name entzerrt
                /in das Deck name komprimiert wer-
                den
```

1. Grundlagen

1.1. Entwurfskriterien

Beim Entwurf des Editors SB&EDIT wurde von den folgenden Ueberlegungen ausgegangen:

- Da umfangreiche Quell-Texte, um Platz auf der Platte zu sparen, am zweckmaessigsten in komprimierter Form abgespeichert werden, sollte Texthaltung sich weitgehend auf die hier schon vorhandene COSY-Struktur beschraenken koennen. Dazu erscheint es sinnvoll, die Textverarbeitung direkt in der COSY-Datei durchzufuehren und eine eventuell notwendige Entzerrung der komprimierten Daten in eine Texthaltungs-Datei nach TR440-Konvention (RAM-Datei mit Satzbau U800 oder aehnlich) nur auf Scratch durchzufuehren, um ein Nebeneinander verschieden weit verarbeiteter Versionen desselben Textes moeglichst zu vermeiden. Die Entzerrung sollte jedoch jederzeit bei Bedarf moeglich sein und auch mit vom Benutzer vorgegebenen Dateien arbeiten koennen.

- Die von COSY angebotene Moeglichkeit, verschiedene Textstuecke in Form von Decks in einer PHYS-Datei halten zu koennen, sollte ausgenuetzt werden koennen, um in einem Editier-Lauf auch mehrere Decks bearbeiten oder etwa miteinander mischen zu koennen.

- Der Datenaustausch mit dem Hintergrund (der COSY-Deck-Datei) sollte in groesseren Informationsbloecken geschehen, wobei es dem Benutzer moeglich sein sollte, auch Teilbloecke zum Hintergrund zurueckzutransportieren und bei Bedarf auch zusaetzliche Bloecke vom Hintergrund zuzuladen.

- Der Umfang dieser Bloecke sollte mehrere Zeilen betragen, um komplexere Editierungsaufgaben weitgehend innerhalb eines Blockes ausfuehren zu koennen; als Block koennte man sich etwa eine (konzeptuelle) Druckseite vorstellen.

- Editierung sollte - mit entsprechendem Daten-/Hintergrund-Transport - auch ueber Blockgrenzen hinweg moeglich sein; dabei sollte es moeglich sein, Blockgrenzen aufzuheben, neu zu erzeugen oder zu verschieben, je nach den Beduerfnissen der aktuellen Editierungsaufgabe.

- Der Zugriff auf bestimmte Textstuecke, um diese zu editieren, sollte ohne Verwendung von Organisationsmitteln wie Zeilen- oder Blocknummern durchgefuehrt werden, da diese im allgemeinen nur zusammen mit einem aktuellen Listing verwendbar sind und sich oft notgedrungen schon waehrend der Editierung selbst so stark aendern, dass die

Editierung bis zum Vorliegen eines neuen Listings unterbrochen werden muss, um Konfusion zu vermeiden.

- Textaenderungen, die mit dem Editor durchgefuehrt werden koennen, sollten unter anderem folgende Funktionen umfassen:
  - Einfuegung von Strings oder einzelnen Zeichen
  - Loeschen von Strings oder Zeichen
  - Ersetzen eines Strings durch einen anderen gleicher oder anderer Laenge
  - Aufsplitten einer Zeile in mehrere
  - Zusammenfuegen mehrerer Zeilen oder Teilzeilen zu einer Zeile

- Die Stelle, an der eine solche Aenderung durchgefuehrt wird, sollte durch den Text selbst unter Verwendung einer Suchanfrage - auch ueber Blockgrenzen hinweg ! - bestimmt werden koennen, um so vom Vorliegen eines aktuellen Listings unabhaengiger zu werden.
  Dabei sollten fuer den zu suchenden String folgende Moeglichkeiten geboten werden:
  - Suche nach exakt einem String
  - Suche unter Ignorierung des Unterschieds zwischen Gross- und Kleinschreibung, eventuell auch nur in einem Teil des Suchstrings
  - Suche unter Ignorierung des Unterschieds einzelner Zeichen/Buchstaben/Ziffern/Trennzeichen/einer Menge vorgegebener Zeichen

- Zusaetzlich sollten die zu aendernden Stellen auch durch explizite (relative und absolute) Positionierung innerhalb des Textes, um beliebig viele Zeichen und/oder Zeilen vorwaerts/rueckwaerts vom Anfang oder Ende, bestimmt werden koennen.

- Es sollte moeglich sein, gleichfoermige Aenderungen, die an mehreren Stellen des Textes durchzufuehren sind, nur ein einziges Mal explizit zu spezifizieren und die Wiederholung der Aenderung durch geeignete Wiederholungsbefehle oder Schleifenoperationen zu veranlassen. Speziell sollte es dabei moeglich sein, denselben Text an beliebig vielen Stellen einzutragen, ohne diesen Text dabei mehr als einmal schreiben zu muessen.

- Es sollte moeglich sein, die Durchfuehrung oder Wiederholung einer Editierungs-Operation vom Ergebnis einer Suchanfrage abhaengig zu machen.

- Der Editor sollte in der Lage sein, den vollen Zeichensatz des Rechners (also auf dem TR440 alle Zeichen des Zentralcodes ZC1) zu bearbeiten. (Diese Forderung liess es sinnvoll erscheinen, sich auf COSY-Decks zu beschraenken, die im Modus K8 oder Z8 erzeugt wurden, also alle ZC1-Zeichen enthalten koennen.)

- Um die Anzahl der Eingaben zum Rechner (und damit die Wartezeiten auf die Antwort vom Rechner) so gering wie moeglich zu halten, sollte der Editor die Spezifikation beliebig vieler und beliebig komplexer Editierungs-Operationen mit einer einzigen Eingabe erlauben.

- Um dem Benutzer unnoetige Schreibarbeit zu ersparen, sollte die Kommandosprache des Editors so aufgebaut sein, dass die Anzahl der einzugebenden Zeichen zur Spezifikation einer bestimmten Editierungsaufgabe minimal ist.

- Die Kommandosprache des Editors sollte einigermassen leicht erlernbar sein, und es sollte moeglich sein, dem Benutzer Hilfsmittel zur Verfuegung zu stellen, etwa in folgender Weise:
  - durch Bereitstellung vorgefertigter Operationen fuer haeufig verwendete Editier-Vorgaenge
  - durch die Moeglichkeit, auch mit geeigneten einfachen Teilmengen des Editors sinnvoll arbeiten zu koennen

- Es sollte moeglich sein, den Editor nicht nur vom Benutzer-Terminal, sondern auch von Quell-Files, also COSY-Decks, die Editor-Kommandos als Text enthalten, zu steuern, etwa vergleichbar dem TUE-Konzept der TR440.

- Die Moeglichkeit von Systemzusammenbruechen sollte beruecksichtigt werden. Fuer diesen Fall ist vorzusehen, die bearbeiteten Dateien zu einem Zustand restaurieren zu koennen, der schlimmstenfalls dem Zustand zu Anfang des aktuellen Editor-Laufes entspricht, moeglichst jedoch wenigstens einen Teil der Ergebnisse dieses Laufes enthaelt.

Ein Editor-System, das von seiner Struktur her einen grossen Teil der vorstehend genannten Entwurfskriterien erfuellt, wurde am MIT von Daniel Murphy unter dem Namen TECO ("Text Editor and Corrector") fuer die pdp-1 entwickelt und von Stewart Nelson und Richard Greenblatt im Rahmen des MIT-Projekts MAC fuer die pdp-6 weiterentwickelt. Da dieses Editorsystem, das inzwischen auch fuer die Rechner pdp-8, pdp-11 und das DEC-System-10 zur Verfuegung steht, sich seit nun fast 10 Jahren recht gut in der Praxis bewaehrt hat und da es dem Benutzer mit (auch geringer) Programmiererfahrung ausserordentlich weitreichende und flexible Moeglichkeiten der Textverarbeitung bietet, schien es sinnvoll, TECO als Grundlage zur Implementierung des Editor-Systems SB&EDIT zu verwenden.

Der recht verschiedene Aufbau der Texthaltung auf der DEC-10 und der TR440 erlaubt es jedoch nicht, TECO unbesehen zu uebernehmen, insbesondere, da zwar der Editor TECO Moeglichkeiten bietet, die ueber die der TR440-Texthaltung weit hinausgehen, dies jedoch nur unter Verwendung stark abweichender und zum Teil zur TR440 inkompatibler Konzepte erreicht. Um Kompatibilitaet mit der bisher verwendeten Texthaltung - unter Einbeziehung der COSY-Dienstleistungen - zu

erhalten, waren somit an einigen Stellen Aenderungen gegenueber dem Original-System vorzunehmen. Diese Aenderungen stellen im wesentlichen TR440-spezifische Erweiterungen oder Ersetzungen dar, die insgesamt jedoch so gering gehalten werden konnten, dass ein Benutzer von TECO ohne nennenswerte Schwierigkeiten auf SB&EDIT umsteigen kann. Dabei ist lediglich zu beachten, dass ein pdp-Textfile hier einem COSY-Deck entspricht und dass die bei TECO verwendeten ASCII-Control-Characters hier durch gewisse Ersatz-Darstellungen repraesentiert werden.

Insbesondere ist das bei TECO als Eingabe-Ende verwendete Doppel-Escape (ESC, Control-[) hier durch das Abschick-Zeichen an den Rechner (im allgemeinen Line-Feed (LF, Control-J) oder Control-D (EOT)) ersetzt. Zeilenwechsel (Carriage-Return, CR, Control-M) wird intern, gemaess TR440-Konvention, auf Line-Feed abgebildet.

Escape (ESC), das bei TECO weitgehend die Funktion des String-Begrenzungs-Zeichens hat, wird hier durch "$" ersetzt, was insofern wohl keine allzugrosse Umstellung ist, als TECO Escape als "$" reflektiert. Ebenso wird der Operator "#" (fuer bitweises OR) durch "$" ersetzt, da das Zeichen "#" bei den meisten Terminals als Fluchtsymbol fuer die TR440 verwendet wird.

Schliesslich werden sonstige Control-Characters (Control-x, wobei x fuer einen beliebigen Grossbuchstaben steht), die bei TECO an vielen Stellen durch die Zeichen-Kombination "^x" ersetzt werden koennen, hier generell durch diese Zeichen-Kombination ersetzt, da eine Uebertragung von Control-Characters direkt im ZVR-System nicht moeglich ist. Diese Aenderungen, die zur Anpassung von TECO an den Betrieb auf der TR440 notwendig waren, sind saemtlich so gegehalten, dass die Umstellung von TECO auf SB&EDIT keine groesseren Schwierigkeiten machen duerfte.

An dieser Stelle sei noch angemerkt, dass die beiden sowohl in TECO als auch in SB&EDIT verwendeten ASCII-Zeichen "^" und "_" auf manchen Bildschirm-Tastaturen als "↑" bzw. "←" dargestellt sind; die Zeichen, die an den Rechner uebertragen werden, sind jedoch bei beiden Darstellungen die gleichen.

1.2. Konzepte

1.2.1. Die Editor-Kommandosprache

Wenn der Editor SB&EDIT durch das Kommando #EDIT gestartet wurde, erwartet er Eingaben des Benutzers, die die auszufuehrenden Editierungsaufgaben und deren Parameter bzw. zu suchende oder einzutragende Texte spezifizieren. Diese Eingaben werden jeweils mit dem Text "Gib Edit-Kommandos:" angefordert. Die Eingabe des Benutzers besteht nun in einer beliebigen formatfreien Folge von Kommandos an den Editor, deren genaue Spezifikation im naechsten Kapitel gegeben ist.

Die Menge aller zulaessigen Folgen von Editor-Kommandos bildet die Editor-Sprache. Eine Eingabe des Benutzers stellt, strenggenommen, ein Programm bzw. einen Programm-Modul in dieser Sprache dar. Dieses Programm kann im einfachsten Fall aus einem einzigen Editor-Kommando bestehen, also einer Anweisung, die sich auf den zu editierenden Text als zugrundeliegende und zu verarbeitende Datenstruktur bezieht. Kontroll-Strukturen in der Editor-Sprache erlauben es, Programme einzugeben, die wesentlich komplexer als einfache lineare Folgen von Kommandos sind. Die folgenden Kontroll-Strukturen stehen zur Verfuegung:

- Bedingungen (entspechend "if - then - fi")
- n-fache Iteration (entsprechend "for - do - od")
- Iteration bis zum Eintritt einer Bedingung (entsprechend "do - od until - ")
- in begrenztem Umfang Spruenge (entsprechend "go to - ")
- Aufruf anderer Editor-Programme als Macros bzw. Unterprogramme (entsprechend "call - "); Aufrufe duerfen geschachtelt sein
- Retten eines Editor-Programms fuer spaetere Verwendung als Macro; Macros koennen dabei vor ihrer Verwendung wie andere Texte editiert und damit veraendert werden, auch unter Verwendung anderer Macros.

Die Sprache ist so aufgebaut, dass Editor-Programme extrem kurz sind - selbst fuer recht komplexe Programme ist selten mehr als eine Zeile noetig - und dass sie bei einiger Uebung fuer nicht allzu komplexe Aufgaben sofort ohne grosse Ueberlegungen hingeschrieben werden koennen. Dies hat allerdings den Effekt, dass unkommentierte Editor-Programme sehr schlecht lesbar sind, was jedoch in Anbetracht des Anwendungsbereiches der Sprache als von untergeordneter Bedeutung anzusehen ist. In dieser Hinsicht laesst sich die Editor-Sprache etwa mit APL vergleichen; sie benoetigt jedoch, im Gegensatz zu APL, kein spezielles Eingabe-Alphabet.

Editor-Programme werden interpretierend abgearbeitet. Ein Fehler im Programm oder Kommandos, die auf den zu editierenden Text nicht anwendbar sind, fuehren zum Abbruch der Abarbeitung, zur Ausgabe einer Fehlermeldung und zur Anfor-

derung einer neuen Eingabe. Als Testhilfen stehen ein Kommando zur Fehlerlokalisierung und die Moeglichkeit, einen Trace bei der Abarbeitung einzuschalten, zur Verfuegung.

Die Abarbeitung des Editor-Programms kann, wie alle normalen Programme auf der TR440, durch das Vermittler-Kommando #XAN unterbrochen werden. Gibt man auf die Meldung ##ABW hin an den Abwickler die Anweisung HALT,SB&EDIT (bzw., wenn der Editor in der Vorrangstufe lauft, die Anweisung HALT,SB&EDIT2), so geht der Editor in einen normierten Zustand ueber und verlangt eine neue Eingabe. Diese Moeglichkeit ist nuetzlich, wenn ein fehlerhaftes Editor-Programm in eine unendliche Schleife geraten ist, da man es so aus dieser Schleife herausholen kann, ohne den Programm-Lauf durch die BEENDE-Anweisung abbrechen zu muessen, was den Verlust aller Daten im Editier-Puffer und auch im Ausgabe-Puffer (der dem Benutzer verborgen ist) zur Folge haette.

Innerhalb eines Programms koennen die einzelnen Kommandos direkt, also ohne Trennzeichen, aufeinander folgen. Man kann jedoch auch beliebig Zwischenraeume (Blanks) und Zeilenwechsel zur optischen Auflockerung und Erhoehung der Uebersichtlichkeit zwischen die Kommandos einfuegen; notwendig ist dies aber nicht.

Die Kommandos bestehen aus ein oder zwei ASCII-Zeichen und koennen zum Teil ein oder zwei numerische Parameter und/oder ein oder zwei String-Parameter haben. Fuer diese Parameter gelten folgende Regeln:

- Numerische Parameter stehen immer vor dem Kommando, auf das sie sich beziehen; wenn ein Kommando zwei numerische Parameter hat, so sind diese durch ein Komma voneinander getrennt.
- Numerische Parameter koennen beliebige arithmetische Ausdruecke sein, deren Operanden durch die (ganzzahligen) Operationen

  "+" Addition
  "-" Subtraktion
  "*" Multiplikation
  "/" integer-Division
  "&" bitweises logisches UND
  "$" bitweises logisches ODER

  miteinander verknuepft sind. Dabei gelten alle Operationen als gleichwertig; die Ausdruecke werden also von links nach rechts abgearbeitet, wenn nicht explizit durch Klammerung eine andere Abarbeitungsreihenfolge vorgeschrieben wird. Operanden koennen ganze Zahlen, Oktalzahlen (die durch ein vorangestelltes "^O" als solche gekennzeichnet sind) und spezielle Sprach-Symbole sein, die entweder bestimmte numerische Werte haben oder denen solche Werte durch bestimmte Editor-Kommandos zugewiesen werden koennen.
- Bestimmte Zeichen bzw. Zeichen-Kombinationen innerhalb der Editor-Sprache koennen als numerische Parameter und

als Operanden in numerischen Parametern eingesetzt werden; diese Symbole haben dabei numerische Werte, die ihnen folgendermassen fest oder variabel (je nach Symbol) zugewiesen sind:
- Werte von Zustands-Schaltern des Editors
- explizit als Variable abgespeicherte numerische Werte
- Positionen im zu editierenden Text
- Code-(ZC1-)Werte von Zeichen
- Zustandsgroessen der COSY-Datei
- Werte von Zahlen im zu editierenden Text

- String-Parameter stehen immer hinter dem Kommando, auf das sie sich beziehen; wenn ein Kommando zwei String-Parameter hat, so muessen beide Strings dasselbe Zeichen als Begrenzungs-Zeichen haben.
- String-Parameter bestehen aus beliebigen Zeichenfolgen, deren erstes Zeichen das Zeichen direkt hinter dem vorangehenden Kommando bzw. hinter dem vorangehenden String-Begrenzungs-Zeichen ist. Der String-Parameter wird entweder durch das Ende der Eingabe oder durch ein spezielles Begrenzungs-Zeichen beendet; dieses ist normalerweise das Zeichen "$". Die Kommandos "I", "^I", "S", "N", "_", "FS" und "FN" erlauben jedoch auch die Verwendung eines beliebigen anderen Zeichens zur Begrenzung ihrer String-Parameter. Dazu ist vor das betreffende Kommando das Zeichen "@" einzufuegen; das erste Zeichen nach dem Kommando wird dann als String-Begrenzungs-Zeichen fuer den folgenden String bzw. bei zwei String-Parametern fuer beide folgenden Strings genommen.
- Spezielle Zeichen innerhalb von String-Parametern erlauben es, diese noch vor ihrer Verwendung durch das zugehoerige Kommando nach Bedarf zu modifizieren.

Bedingte Kommandos und Iterationen werden durch Paare von Klammern dargestellt; dabei muessen die zusammengehoerenden Klammern innerhalb jeder Ebene der Macro-Verschachtelung vollstaendig sein.

Die Sprache verfuegt ueber zwei Saetze von Software-Registern, die in Anlehnung an die Terminologie von TECO als "Q-Register" bezeichnet werden. Jeder dieser Saetze umfasst 36 Register. Die Register des ersten Satzes werden durch die 10 Ziffern und die 26 Buchstaben adressiert; die Register des zweiten Satzes sind als Stack organisiert; auf sie kann nur ueber Register des ersten Satzes zugegriffen werden. Jedes Register kann entweder einen numerischen Wert oder einen String beliebiger Laenge speichern. Enthaelt ein Register einen String, so kann dieser insbesondere ein Editor-Programm sein, das dann als Macro aufrufbar ist; die Q-Register werden somit auch zur Implementierung der Unterprogramm-Faehigkeiten der Editor-Sprache verwendet.

Zum Abschluss der Sprachbeschreibung sei noch bemerkt, dass bei der hier beschriebenen Version 2 der Editor-Sprache (im

Gegensatz zur Version 1) grosse und kleine Buchstaben als Kommandos und Bezeichnungen numerischer Werte aequivalent sind; dies gilt natuerlich nicht fuer die Texte innerhalb von String-Parametern. In der folgenden Beschreibung des Editors wird jedoch bei der Angabe von Kommandos grundsaetzlich Grossschreibung verwendet, waehrend zur Unterscheidung Bezeichnungen fuer variabel einzusetzende Parameter von Editor-Kommandos klein geschrieben werden.

1.2.2. Text-Files

Der Editor SB&EDIT bearbeitet Texte, die in Einheiten zusammengefasst sind, die als "File" bezeichnet werden. Auf dem Hintergrund entspricht ein solcher File genau einem Deck in der bearbeiteten COSY-Deck-Datei. Der Editor arbeitet mit bis zu zwei Files gleichzeitig, hat also bis zu zwei Decks, die in derselben COSY-Datei liegen muessen, in der Bearbeitung.

Ein Deck kann auf zwei Arten vom Editor zur Bearbeitung eroeffnet werden:
- als Eingabe-File, aus dem der Editor Text entnehmen kann, das er aber nicht veraendern kann. Jedes beliebige Deck in der COSY-Datei (ausser im Modus K6 erzeugten Decks) kann Eingabe-File fuer den Editor werden.
- als Ausgabe-File, in den der Editor Text eintragen kann, aus dem er aber, solange das Deck Ausgabe-File ist, keinen Text entnehmen kann. Ein Deck, das Ausgabe-File werden soll, wird vom Editor als leeres Deck erzeugt, falls die COSY-Datei noch nicht zu viele Decks enthaelt (maximal 49 Decks sind in einer COSY-Datei erlaubt).

Ein Editierlauf eroeffnet ueblicherweise ein COSY-Deck als Eingabe-File (etwa durch den Befehl "ER"), erzeugt ein zweites, neues Deck als Ausgabe-File (etwa durch den Befehl "EW"), und fuehrt die Editierung durch, indem Information vom Eingabe-File gelesen wird, gegebenenfalls geaendert wird und schliesslich in den Ausgabe-File geschrieben wird.

Es ist dabei durchaus moeglich, den Eingabe-File waehrend der Editierung zu wechseln; dazu ist es lediglich noetig, ein neues Deck als Eingabe-File zu eroeffnen. Diese Eigenschaft des Editors erlaubt es zum Beispiel, mehrere Decks miteinander zu mischen. Ebenso kann man auch den Ausgabe-File waehrend eines Editierlaufes wechseln; hierzu ist es jedoch erforderlich, zuerst den alten Ausgabe-File explizit (durch den Befehl "EF") zu schliessen, ehe ein neues Deck als Ausgabe-File erzeugt werden kann.

In vielen Faellen ist es erwuenscht, Editierung als Aenderung eines vorhandenen Textes durchzufuehren, wobei der Text nach Abschluss des Editiervorgangs in geaenderter Form, aber unter dem alten Namen, zur Verfuegung stehen sollte. Hierzu bietet der Editor die Dienstleistung der sogenannten Back-up-Editierung an, bei der nach Abschluss des Editiervorgangs zusaetzlich noch die alte Text-Kopie zur Verfuegung steht, was einen gewissen Schutz gegenueber Systemzusammenbruechen waehrend des Editiervorgangs bietet. Die Back-up-Editierung laeuft im Einzelnen folgendermassen ab:
- Das zu aendernde Deck wird als Eingabe-File zum Lesen eroeffnet.
- Ein leeres Deck mit dem (reservierten) Namen TEMP&BACK&UP

wird als Ausgabe-File erzeugt.
- Die eigentliche Editierung laeuft wie ueblich ab, indem vom Eingabe-File gelesen wird und der (geaenderte) Text in den Ausgabe-File geschrieben wird.
- Wenn der Ausgabe-File durch einen der Befehle "EF", "^Z", "EX" oder "EL" geschlossen wird, erhaelt er den Namen des aktuellen Eingabe-Files, und dieser erhaelt den (ebenfalls reservierten) Namen EDIT&BACK&UP.
- Als Sicherheiten gegen Fehlbedienungen des Back-up-Mechanismus gelten folgende zusaetzliche Regeln:
  - Wenn aufgrund eines Zusammenbruchs oder durch gewaltsamen Programm-Abbruch der Editiervorgang nicht ordnungsgemaess abgeschlossen wird, steht der alte Text unter seinem alten Namen zur Verfuegung. Soweit geaenderte Information schon ausgeschrieben wurde, steht sie im Deck TEMP&BACK&UP.
  - Die Back-up-Editierung kann nicht gestartet werden, wenn ein Deck mit dem Namen TEMP&BACK&UP existiert, da dies auf einen abgebrochenen Editiervorgang hindeutet, dessen Ergebnisse erst gerettet werden sollten, wozu aber explizite Deklaration der Ein- und Ausgabe-Files erforderlich ist (durch die Kommandos "ER" bzw. "EW").
  - Es wird jeweils maximal eine Back-up-Kopie gehalten; wenn bei Beginn der Back-up-Editierung schon ein Deck mit dem Namen EDIT&BACK&UP existiert, wird es kommentarlos geloescht.

Durch den Parameter DECK im EDIT-Kommando kann schon beim Start des Editors ein COSY-Deck zur Bearbeitung eroeffnet werden. Wenn dieser Parameter besetzt ist, sucht der Editor in der COSY-Datei, ob schon ein Deck mit dem angegebenen Namen existiert. Falls dies der Fall ist, wird es als Eingabe-File zur Back-up-Editierung eroeffnet, und der erste Text-Block wird eingelesen. (Es wird also die Befehlsfolge "EB", "Y" simuliert.) Wenn noch kein Deck mit dem angegebenen Namen existiert, wird ein leeres Deck mit diesem Namen als Ausgabe-File erzeugt. (In diesem Fall wird somit beim Start des Editors der Befehl "EW" simuliert.)

Nicht mehr benoetigte Decks koennen durch das Kommando "ED" geloescht werden; der Platz dieser Decks wird dabei freigegeben, so dass die COSY-Datei durch ein anschliessendes RESERVIERE-Kommando verkleinert werden kann.

Jedem Deck ist eine Nummer zugeordnet, und zwar sind die Decks der COSY-Datei in ihrer physikalischen Reihenfolge von 1 an durchnumeriert. Die Nummer eines Decks ist diesem somit nicht unveraenderbar zugeteilt, sondern kann sich durch Loeschen eines davorliegenden Decks erniedrigen; ausserdem veraendert der Back-up-Mechanismus natuerlich die Nummer des bearbeiteten Decks und moeglicherweise noch anderer Decks.

Man hat in der Kommando-Sprache des Editors in gewissem Umfang Zugriff auf Information der COSY-Deck-Struktur:

- Durch den Befehl "OEA" kann man sich eine Liste aller Decks der COSY-Datei ausgeben lassen.
- Das Symbol "EA" hat als numerischen Wert die Gesamtanzahl der Decks in der Datei.
- Das Symbol "EN" hat als numerischen Wert die Nummer des Eingabe-Files bzw. 0, wenn es keinen Eingabe-File gibt.
- Der Name des n-ten Decks kann durch den Befehl "nEA" in den Text eingefuegt werden.

Schliesslich kann man durch das Kommando "EI" ein Deck als Eingabe-File eroeffnen, den darin enthaltenen Text einlesen und als Folge von Editor-Kommandos ausfuehren lassen. Diese Dienstleistung des Editors entspricht somit in gewissem Sinne dem TUE-Kommando der TR440-Kommandosprache.

1.2.3. Datentransport und Editier-Puffer

Text, der editiert werden soll, wird vom Hintergrund, also von dem COSY-Deck, das als Eingabe-File eroeffnet ist, in einen Editier-Puffer eingelesen, dort gegebenenfalls veraendert und schliesslich auf den Hintergrund, also das COSY-Deck, das als Ausgabe-File erzeugt wurde, geschrieben. Die Editor-Sprache stellt zu diesem Zweck Kommandos zur Verfuegung, die es gestatten, Text einzulesen und auszuschreiben. Der Benutzer hat dabei keine Moeglichkeit, die Stelle, von der gelesen bzw. auf die geschrieben werden soll, zu spezifizieren, da diese Stellen durch die COSY-Struktur schon festgelegt sind. Einlesen und Ausschreiben geschehen rein sequentiell innerhalb der bearbeiteten Decks. Entsprechend bestehen die zugehoerigen Kommandos nur aus Befehlen, die Lesen und Schreiben und z.T. davon betroffene Teile des Editier-Puffers angeben; um die Organisation der Daten auf dem Hintergrund braucht der Benutzer sich nicht zu kuemmern. Ebenso bleibt dem Benutzer die komprimierte Darstellung des Textes auf dem Hintergrund verborgen; beim Einlesen wird der Text automatisch entzerrt, und beim Ausschreiben wird er automatisch komprimiert.

Diese Form des Hintergrund-Transportes hat zur Folge, dass bei jedem Editor-Lauf solange Daten vom Eingabe-File gelesen werden, bis die sich die erste zu editierende Text-Stelle im Editier-Puffer befindet. Da fuer die Anzahl der Zeichen, die normalerweise im Puffer gehalten wird, eine obere Schranke existiert - normalerweise werden etwa 5000 Zeichen im Puffer gehalten - kann es sein, dass auch vor Erreichen dieser ersten zu editierenden Stelle schon Text in den Ausgabe-File geschrieben wird. Man kann sich vorstellen, dass der zu editierende Text vom Editor in Seiten von etwa 5000 Zeichen unterteilt wird. Der Editor blaettert nun diese Seiten von vorne an der Reihe nach durch - holt sie in den Puffer -, veraendert sie moeglicherweise und blaettert weiter - schreibt sie in den Ausgabe-File.

Wichtig ist hierbei, dass es nicht moeglich ist, zurueckzublaettern, also zu einer schon auf den Ausgabe-File ausgeschriebenen Seite zurueckzukehren, ohne einen neuen Durchgang durch das zu editierende Deck zu beginnen. Dieser Nachteil wird dadurch ausgeglichen, dass der Editier-Puffer in seiner Groesse variabel ist: Es ist ohne weiteres moeglich, zu dem darin vorhandenen Text an beliebiger Stelle - auch innerhalb einer Zeile ! - beliebige Mengen neuen Textes hinzuzufuegen. Dieser zusaetzliche Text kann aus dem Eingabe-File, aus einem Q-Register oder aus einem expliziten Eintrage-Kommando entnommen werden. Ebenso kann der Puffer auch durch Loesch-Befehle verkuerzt werden. In jedem Falle werden solche Aenderungen der Puffer-Laenge jedoch durchgefuehrt, ohne dass dabei automatisch Daten-Transporte vom oder zum Hintergrund durchgefuehrt werden.

Daten werden vom Hintergrund eingelesen, wenn einer der Befehle "Y" oder "A" gegeben wird; der Unterschied zwischen diesen beiden Befehlen besteht darin, dass "Y" vor dem Einlesen den Puffer loescht - dieser Befehl eignet sich also besonders fuer erstmaliges Einlesen -, waehrend "A" den neuen Text an den schon im Puffer vorhandenen einfach hinten anhaengt. Beim ersten Einlesen in einen leeren Puffer werden etwa 5000 Zeichen eingetragen, wobei die Datenuebertragung nach Moeglichkeit bei einem Zeilenwechsel beendet wird; bei weiterem Einlesen mittels "A" wird etwa ein Block der COSY-Datei (nach dem Entzerren etwa 1000 Zeichen) uebertragen, wobei auch hier normalerweise bei einem Zeilenwechsel aufgehoert wird.

Ausschreiben der Daten muss explizit durch einen Befehl "P" oder "PW" veranlasst werden; wenn keiner dieser Befehle gegeben wird, werden keine Daten in den Ausgabe-File uebertragen, speziell auch dann nicht, wenn der Ausgabe-File geschlossen wird. Der Befehl "PW" gibt dabei nur den aktuellen Puffer aus, veraendert aber sonst nichts; insbesondere bleibt der Puffer-Inhalt erhalten. Der Befehl "P" gibt, wenn er zwei Parameter hat, den durch diese Parameter bestimmten Teil des Puffers aus, laesst aber sonst ebenfalls den Puffer unveraendert. Dagegen gibt der Befehl "P", wenn er nur einen oder gar keinen Parameter hat, den ganzen Puffer aus, loescht den Puffer und liest eine neue Seite ein - dieser Befehl fuehrt also genau das oben beschriebene Umblaettern der einzelnen Seiten des Textes durch, wobei man auch spezifizieren kann, wieviele Seiten auf einmal umgeschlagen werden sollen.

Da es muehevoll ist, bei einer Suche durch einen laengeren Text dessen Unterteilung in Seiten beachten zu muessen und die Seiten dabei immer explizit austauschen zu muessen, besteht die Moeglichkeit, Suche nach Text-Strings ueber Seitengrenzen hinweg zu spezifizieren. Bei einer solchen Suche werden solange automatisch Seiten ausgetauscht, bis entweder der gesuchte Text gefunden ist oder das Ende des Eingabe-Files erreicht wurde.

Die im Editier-Puffer stehende Textmenge wird als linearer Zeichen-String aufgefasst; dieser String kann sich ueber beliebig viele Zeilen erstrecken, wobei die einzelnen Zeilen durch LF-(Line-Feed-) bzw. FF-(Form-Feed-)Zeichen voneinander getrennt sind. Der Text enthaelt sonst keinerlei Struktur-Information; speziell gibt es keine Numerierung der Zeilen im Text. Dies hat insbesondere den Vorteil, dass man zwei benachbarte Zeilen durch einfaches Loeschen des sie trennenden Zeilenwechsel-Zeichens zu einer einzigen zusammenfassen und dass man eine Zeile durch Einfuegen von Zeilenwechsel-Zeichen in mehrere aufbrechen kann.

Jeder Position im Editier-Puffer ist eine nicht-negative, ganze Zahl zugewiesen, und zwar ist diese Zahl gleich der

Anzahl der Zeichen im Puffer, die sich vor dieser Stelle befinden. Eine solche Puffer-Position weist also genaugenommen immer zwischen zwei Zeichen im Puffer, oder vor das erste oder hinter das letzte Zeichen.

Es gibt nun einen Zeiger, mit dem man die Stelle im Puffer, an der man die naechste Editierungs-Operation durchfuehren will, markieren kann. Dieser Zeiger wird auf eine bestimmte Position im Puffer eingestellt, indem man ihm den Zahlenwert zuweist, der dieser Position entspricht. Die verschiedenen Moeglichkeiten, diese Zuweisung durchzufuehren, werden im uebernaechsten Abschnitt erlaeutert.

Bestimmte Symbole in der Editor-Sprache repraesentieren Puffer-Positionen, die haeufig eingestellt werden muessen:
- "B" hat den Wert 0, repraesentiert also den Puffer-Anfang
- "Z" hat als Wert die Anzahl der gerade im Puffer befindlichen Zeichen, repraesentiert also das Puffer-Ende
- "." hat als Wert die Anzahl der Zeichen vor dem Zeiger, repraesentiert also den Zeiger
- "H" repraesentiert das Paar von Werten "B,Z", kann also bei Befehlen mit zwei Argumenten verwendet werden, um den ganzen Puffer als Parameter anzugeben.

Diese Symbole werden in der Editor-Sprache ueberall dort als Parameter von Editor-Kommandos eingesetzt, wo die entsprechenden Zahlenwerte zur Spezifikation irgendwelcher Positionen im Editier-Puffer benoetigt werden.

Nach Abschluss des letzten Editier-Vorganges fuer den zu bearbeitenden Text muss noch der Rest dieses Textes ausgegeben werden und der Ausgabe-File geschlossen werden. Dies kann etwa dadurch geschehen, dass durch hinreichend viele "P"-Kommandos der aktuelle Editier-Puffer ausgegeben wird und solange vom Eingabe-File weitere Seiten eingelesen und wieder ausgeschrieben werden, bis dieser erschoepft ist. Anschliessend kann der Ausgabe-File durch ein Kommando "EF" geschlossen werden, bzw. kann man mit dem Kommando "^Z" den Ausgabe-File schliessen und anschliessend den Editor-Lauf beenden.

Einfacher ist es jedoch, den Editor-Lauf direkt mit dem Kommando "EX" zu beenden. Dieses Kommando fuehrt das Umkopieren des restlichen Eingabe-Files auf den Ausgabe-File automatisch aus und beendet den aktuellen Editor-Lauf; der Ausgabe-File enthaelt dann den fertig editierten, vollstaendigen Text.

1.2.4. Ausgabe auf Texthaltungs-Dateien

Die vom Editor erzeugten Ausgabe-Files werden in der COSY-Datei als Decks im Modus K8 (auch wenn der Eingabe-File ein Deck im Modus Z8 war!) gespeichert.

Wenn man den Inhalt dieser Decks fuer die weitere Verwendung durch das Programmier-System der TR440 zur Verfuegung stellen will, ist es erforderlich, den darin enthaltenen Text in entzerrter Form in eine Texthaltungsdatei nach TR440-Konvention einzutragen.

Dies kann auf verschiedene Weise erreicht werden:
- Man kann das Dienstprogramm SB&COSY zum Entzerren der benoetigten Decks im Modus E bzw. E8 einsetzen. Die Rechenzeit, die zum Entzerren eines Decks benoetigt wird, ist dabei vergleichbar mit der Zeit, die zum Numerieren einer gleichgrossen Texthaltungsdatei mittels des Kommandos TNUMERIERE gebraucht wird.
- Man kann die Kommandos "EP" und "EL" des Editors zum Entzerren verwenden, wenn der Parameter AUSGABE des Kommandos EDIT besetzt war. Die benoetigte Rechenzeit ist vergleichbar mit der im ersten Falle.

Das Kommando "EP" eroeffnet ein Deck der COSY-Datei als Eingabe-File, liest eine Seite nach der anderen in den Editier-Puffer und schreibt die einzelnen Zeilen im Puffer nach TR440-Texthaltungskonvention in eine Ausgabe-Datei oder ein Ausgabe-Protokoll. Dabei werden Zeilen mit mehr als 160 Zeichen nach jeweils 160 Zeichen aufgebrochen, so dass nie Zeilen mit mehr als 160 Zeichen in eine Texthaltungsdatei oder in das Ausgabe-Protokoll eingetragen werden. Dabei geht natuerlich keine Information verloren, aber die Struktur des Textes ist geaendert, wenn die Laenge der Zeilen fuer diese Struktur wesentlich ist. Zeilen mit maximal 160 Zeichen werden mit ihrer echten Laenge ausgegeben. Das Kommando "EL" wirkt zunaechst wie das Kommando "EX", schliesst aber, ehe der Editor-Lauf beendet wird, noch einen Entzerrungs-Lauf entsprechend dem Kommando "EP" durch den Ausgabe-File des vorangegangenen Editier-Laufes an. Wenn der Editor durch das Kommando "EL" verlassen wird, steht somit im allgemeinen sofort eine entzerrte Version des soeben editierten Textes zur Verfuegung.

Die beiden Parameter AUSGABE und PROTOKOLL des Kommandos #EDIT bestimmen dabei, wohin das durch die Editor-Kommandos "EP" und "EL" bearbeitete COSY-Deck entzerrt wird:

- Ausgabe in eine Texthaltungsdatei erfolgt nur dann, wenn der Parameter AUSGABE besetzt ist, und zwar:
  - Wenn fuer AUSGABE ein Datei-Name angegeben ist, so erfolgt die Ausgabe des entzerrten Textes in die angegebene Datei; falls diese noch nicht existiert, wird sie

automatisch in der angegebenen Datenbasis erzeugt (bzw. wenn keine Datenbasis angegeben ist, in der Standard-datenbasis), andernfalls wird vor der Ausgabe der alte Inhalt der Datei geloescht.
  - Wenn der Parameter AUSGABE mit dem Wert -STD- besetzt ist - dies ist die Voreinstellung -, so liegt die Ausgabedatei in der Standarddatenbasis und hat den Namen des entzerrten Decks bzw. wird dort mit diesem Namen erzeugt, wenn es in der Standarddatenbasis keine Datei mit diesem Namen gibt.
  - Wenn die Datei fuer Ausgabe in entzerrter Form waehrend der Ausgabe zu klein wird, so wird sie automatisch vergroessert, sofern Platz ist; andernfalls wird dann die Ausgabe abgebrochen und die Bearbeitung der Ausgabedatei beendet.

- Ein Protokoll der Ausgabe wird nur erstellt, wenn der Parameter PROTOKOLL im Kommando #EDIT besetzt ist, und zwar:
  - Wenn der Parameter PROTOKOLL mit dem Wert -STD- besetzt ist, erfolgt Protokollierung in Standard-Numerierung ins Druckerprotokoll, sofern dieses eingeschaltet ist.
  - Wenn der Parameter PROTOKOLL mit dem Wert KO besetzt ist, erfolgt die Protokollierung zusaetzlich auf das Terminal des Benutzers.

1.2.5. Zeiger-Positionierung und Loeschen

Wenn der im Editier-Puffer befindliche Text veraendert werden soll, so geschieht dies bei den meisten Editor-Kommandos an der Stelle, auf die der Puffer-Zeiger gerade weist. Ein typischer Editiervorgang besteht somit aus 2 Schritten:
- Zuerst wird der Zeiger auf die Stelle im Puffer gesetzt, die veraendert werden soll.
- Dann wird mit einem weiteren Editor-Kommando die Stelle im Puffer, auf die der Zeiger weist, in der gewuenschten Weise behandelt.

Es gibt im wesentlichen 4 verschiedene Moeglichkeiten, den Puffer-Zeiger auf eine bestimmte Stelle im Puffer zu setzen:
- absolute Positionierung
- relative Positionierung um eine bestimmte Anzahl von Zeichen
- relative Positionierung auf einen bestimmten Zeilenanfang
- kontextbezogene Positionierung

- Der Zeiger kann durch den Befehl "nJ" hinter das n-te Zeichen im Puffer gesetzt werden; dabei ist n ein numerischer Parameter, wie im Abschnitt 1.2.1. beschrieben. Absolute Positionierung wird meist verwendet, um den Zeiger auf den Pufferanfang - "0J" oder einfach "J" -, auf das Pufferende - "ZJ" - oder auf eine Position, deren numerischer Wert in einem Register i gespeichert ist, - "QiJ" - zu setzen.

- Relative Positionierung geschieht von der aktuellen Zeiger-Position aus um n Zeichen vorwaerts - "nC" - oder rueckwaerts - "nR" -; sie kann zur Fein-Positionierung um wenige Zeichen verwendet werden. Dabei ist zu beachten, dass "C" aequivalent mit "1C", "R" mit "1R" und "nR" mit "-nC" ist.

- Zeilenbezogene Positionierung geschieht in analoger Weise durch den Befehl "nL", der den Puffer-Zeiger von der aktuellen Position aus auf den Anfang der n-ten Zeile von dieser Stelle aus schiebt. Dabei bedeutet "0L" Positionierung auf den Anfang der Zeile, in der der Zeiger sich gerade befindet, "L" oder "1L" Positionierung auf den Anfang der naechsten Zeile, "-L" oder "-1L" Positionierung auf den Anfang der vorhergehenden Zeile. Generell gilt, dass der Befehl "nL" bei positivem n vorwaerts, andernfalss rueckwaerts positioniert. Das Zeilenwechsel-Zeichen, das zwei benachbarte Zeilen voneinander trennt, wird dabei jeweils als zu der vorangehenden Zeile gehoerig betrachtet. Bei zeilenorientierter Positionierung steht der Zeiger also immer direkt hinter einem Zeilenwechsel-Zeichen und vor dem ersten Zeichen der Zeile, auf die positioniert wurde.

- Kontext-bezogene Positionierung des Zeigers geschieht dadurch, dass mit einem Suchbefehl ein Textstueck - der (erste) String-Parameter dieses Befehls - spezifiziert wird. Ausgehend von der aktuellen Zeiger-Position wird dann der Puffer nach dem (n-ten) Auftreten des spezifizierten Textes durchsucht. Wenn der Text gefunden wurde, steht der Zeiger anschliessend direkt hinter dem letzten Zeichen des Textes im Puffer. Die Reaktion in dem Fall, dass das Ende des Puffers erreicht wurde, ohne dass der gesuchte String gefunden wurde, haengt von der Form des Suchkommandos ab:
  - Bei Suche mit dem Kommando "S" wird eine Fehlermeldung ausgegeben.
  - Bei Suche mit dem Kommando "_" wird der Puffer geloescht, ein neuer Puffer eingelesen und die Suche dort fortgesetzt. Dieses Verfahren wird solange fortgesetzt, bis entweder der Text gefunden wurde oder bis der Eingabe-File erschoepft ist. Das Such-Kommando "_" kann somit dazu verwendet werden, um groessere Teile des Eingabe-Files ohne Erzeugen von Ausgabe-Daten zu ueberspringen.
  - Das Such-Kommando "N" wirkt wie das Kommando "_", doch wird jeweils der aktuelle Puffer in den Ausgabe-File ausgeschrieben, ehe ein neuer Puffer eingelesen wird. Dieses Kommando laesst sich somit am besten dazu verwenden, die naechste zu bearbeitende Stelle im Text zu lokalisieren, wobei die uebersprungenen Seiten unveraendert vom Eingabe-File in den Ausgabe-File uebertragen werden.
  - Die Such-Kommandos koennen durch Voransetzen des Zeichens ":" so modifiziert werden, dass sie einen Zahlenwert als Ergebnis der Suche zurueckbringen, und zwar den Wert -1, wenn die Suche erfolgreich war, andernfalls den Wert 0. Dieser Wert kann als Operand in numerischen Parametern weiterverwendet werden. Wenn die Suche nicht erfolgreich war, wird in diesem Fall keine Fehlermeldung ausgegeben.

  Die kontext-bezogene Editierung mithilfe der Such-Kommandos stellt die zweckmaessigste und bequemste Art der Positionierung innerhalb des Textes dar; sie wird daher die Form der Positionierung sein, die man zur grossraeumigen Lokalisierung der zu bearbeitenden Textstellen verwendet. Da es oft vorkommt, dass man einen so lokalisierten Text durch einen anderen ersetzen will, gibt es zwei weitere Such-Kommandos, die diese Ersetzung direkt an eine erfolgreiche Suche anschliessen:
  - Das Kommando "FS", das zwei String-Parameter hat, verwendet den ersten als String, nach dem in derselben Weise gesucht wird, wie dies beim Kommando "S" der Fall ist. Wenn der gesuchte String im Puffer gefunden wurde, so wird er durch den String ersetzt, der als zweiter String-Parameter des Kommandos angegeben ist; wenn dieser zweite Parameter aus dem leeren String besteht, so wird der gesuchte String lediglich aus dem Puffer geloescht, ohne dass dafuer ein neuer String eingesetzt

wird.
- Das Kommando "FN" wirkt in derselben Weise, doch werden hier automatisch Seitentransporte wie beim Kommando "N" solange durchgefuehrt, bis der gesuchte String gefunden ist oder das Ende des Eingabe-Files erreicht ist.

Beim Einlesen einer neuen Seite wird der Pufferzeiger auf den Pufferanfang gesetzt, ausser wenn mit dem Kommando "A" diese Seite an den im Puffer vorhandenen Text hinten angehaengt wird; in diesem einen Fall bleibt der Wert des Zeigers unveraendert. Beim Einfuegen irgenwelcher Textstuecke in den Puffer - diese Einfuegung geschieht immer an der Stelle, auf die der Zeiger weist - wird der Zeiger um die Anzahl der neu eingefuegten Zeichen erhoeht, steht also nach dem Einfuegen hinter dem neu eingefuegten Text. Diese Eigenschaft des Zeigers erlaubt es, mit aufeinanderfolgenden Einfuegungs-Kommandos fortlaufend Text in den Puffer einzutragen, aehnlich wie man durch das SCHREIBE-Kommando der TR440 fortlaufend neuen Text an eine Texthaltungsdatei anhaengen kann; doch kann, im Gegensatz zum SCHREIBE-Kommando, der Text an jeder Stelle des Puffers eingetragen werden, also auch irgendwo in der Mitte, sogar innerhalb einer Zeile.

Wenn Text aus dem Puffer geloescht wird, so bleibt der Wert des Zeigers unveraendert, wenn der geloeschte Text hinter dem Zeiger steht. Wenn der geloeschte Text vor dem Zeiger steht, wird der Wert des Zeigers um die Anzahl der geloeschten Zeichen vermindert, so dass der Zeiger auch dann nach dem Loeschen noch auf dieselbe Position im Text weist. Wenn beiderseits des Zeigers Text geloescht wird, so wird der Zeiger nur um die Anzahl der geloeschten Zeichen links vom Zeiger vermindert, so dass der Zeiger nachher in die Luecke weist. Hier sei noch darauf hingewiesen, dass beim Loeschen von Text aus dem Puffer dort kein "Loch" entsteht; der Text hinter der geloeschten Stelle wird dicht an den Text vor dieser Stelle herangeschoben. Der Zeiger macht also diese Verschiebungen genau mit, wenn er davon betroffen ist, also auf einen der verschobenen Textteile weist.

Zum Loeschen von Text aus dem Puffer stehen die folgenden Kommandos zur Verfuegung:
- Das Kommando "nD" loescht n Zeichen direkt hinter dem Zeiger, wenn n positiv ist, bzw. n Zeichen direkt vor dem Zeiger, wenn n negativ ist. "D" entspricht dabei "1D", "-D" entspricht "-1D", "0D" ist ohne Wirkung.
- Das Kommando "m,nK" loescht alle Zeichen zwischen den Pufferpositionen m und n, also das m+1-te bis zum n-ten Zeichen im Puffer; speziell loescht "HK" den ganzen Puffer. Dieses Kommando loescht also Text unabhaengig von der aktuellen Position des Zeigers; doch kann dadurch, wie oben beschrieben, der Zeiger als Folge der Loeschoperation mit dem hinter der geloeschten Stelle stehenden Text verschoben werden.

- Das Kommando "nK" loescht alle Zeichen von der aktuellen Position des Zeigers bis zum Anfang der n-ten Zeile, gerechnet von dieser Stelle; dabei werden die Zeilenanfaenge hinter dem Zeiger durch positives n bezeichnet, waehrend im anderen Falle Zeilenanfaenge vor dem Zeiger gezaehlt werden, analog wie beim Befehl "nL". Speziell wird durch den Befehl "0K" der Anfang der Zeile, in der der Zeiger steht, bis zur Position des Zeigers geloescht; dagegen loescht der Befehl "K" oder "1K" alle Zeichen vom Zeiger bis zum Ende der Zeile, in der der Zeiger steht, einschliesslich des die Zeile abschliessenden Zeilenwechsel-Zeichens.

1.2.6. Protokollierung von Texten und numerischen Werten

Der Editor stellt eine Reihe von Kommandos zur Verfuegung, mit denen man beliebige Teile des Editier-Puffers, die Werte numerischer Parameter, beliebige Texte, den Inhalt von Q-Registern sowie die Deck-Struktur der COSY-Datei protokollieren lassen kann. Im Einzelnen handelt es sich dabei um die folgenden Kommandos:

- Das Kommando "m,nT" gibt alle Zeichen zwischen den Positionen m und n im Editier-Puffer aus, also das m+1-te bis zum n-ten Zeichen; speziell gibt "HT" den ganzen Puffer aus.

- Das Kommando "nT" gibt alle Zeichen zwischen der aktuellen Position des Puffer-Zeigers und dem n-ten Zeilenwechsel, von dort gezaehlt, aus; dabei werden bei positivem n die Zeilenwechsel vom Zeiger nach hinten gezaehlt, andernfalls nach vorn (wie bei den Befehlen "nL" und "nK"). Speziell gibt der Befehl "0T" alle Zeichen der Zeile, in der der Zeiger steht, vom Zeilenanfang bis zum Zeiger aus, der Befehl "T" oder "1T" alle Zeichen dieser Zeile vom Zeiger bis zum Zeilenende. Die Zeile, in der der Zeiger steht, kann somit durch die Befehlsfolge "0TT" ganz ausgegeben werden. Der Befehl "-T" entspricht "-1T".

- Der Wert eines numerischen Parameters kann durch das Kommando "=" als Dezimal-Zahl und durch das Kommando "==" als Oktalzahl ausgegeben werden.

- Wenn ein Q-Register i Text enthaelt, so kann dieser durch den Befehl "^Bi" ausgegeben werden.

- Ein beliebiger Text "text" kann ausgegeben werden, indem man ihn zwischen zwei Zeichenpaare "^A" einschliesst, in der Form "^Atext^A".

- Durch den Befehl "^L" kann man im Ablaufprotokoll einen Vorschub auf eine neue Druckseite erzeugen.

- Schliesslich kann man sich durch den Befehl "0EA" eine Liste aller Decks der bearbeiteten COSY-Datei ausgeben lassen.

Der Editor verfuegt ueber gewisse Zustands-Schalter, mit denen sich die Form dieser Protokollierung - nicht aber die Ausgabe eines entzerrten COSY-Decks ueber die Parameter AUSGABE und PROTOKOLL des Kommandos #EDIT ! - den aktuellen Erfordernissen anpassen laesst:

- Der Schalter "ET" bestimmt die Form, in der nicht druckbare ASCII-Control-Characters ausgegeben werden:
  - Wenn der Wert des Schalters 1 ist, so werden diese Zei-

chen unveraendert ausgegeben, was gemaess TR440-Konven-
im allgemeinen zur Ausgabe von Ausrufezeichen fuehrt.
Diese Form der Ausgabe ist die vom Editor angenommene,
wenn der Schalter nicht explizit anders gesetzt wird.
- Wenn der Schalter den Wert 0 hat, so wird ein nicht druckbares ASCII-Control-Character "Control-x" durch die Ersatz-Darstellung "^x"ausgegeben. Eine Ausnahme bildet dabei das Horizontal-Tabulator-Zeichen HT (Control-I), das dann durch einen Software-Tabulator ersetzt wird; es werden bis zur naechsten durch 8 teilbaren Position in der Zeile Blanks als Ersatz-Darstellung des Zeichens HT ausgegeben. Da diese Moeglichkeit zum Aufbau von Tabellen ganz praktisch ist, wird diese Ersatz-Darstellung von HT (als Einzige!) auch in die Ausgabe auf Texthaltungsdatei durch die Kommandos "EP" und "EL" uebernommen, wenn der Wert des Schalters "ET" 0 ist.

- Der Schalter "EU" bestimmt die Form, in der Buchstaben ausgegeben werden:
  - Wenn der Wert des Schalters -1 ist, so werden alle Buchstaben unveraendert ausgegeben. Diese Form der Ausgabe wird vom Editor angenommen, wenn der Schalter nicht explizit anders gesetzt wird.
  - Wenn der Wert des Schalters 0 ist, werden kleine Buchstaben durch den entsprechenden Grossbuchstaben mit vorangesetztem Apostroph ausgegeben, also etwa "a" als "'A". Diese Form der Ausgabe kann nuetzlich sein, wenn man an einem Terminal arbeiten muss, das nur Grossschreibung hat.
  - Wenn der Wert des Schalters 1 ist, werden grosse Buchstaben durch den entsprechenden Kleinbuchstaben mit vorangesetztem Apostroph ausgegeben, also zum Beispiel "A" als "'a".

- Der Wert des Schalters "ES" bestimmt, ob und wie die bei Such-Kommandos gefundenen Zeilen automatisch protokolliert werden:
  - Wenn der Wert des Schalters 0 ist, werden gefundene Zeilen nicht protokolliert; dies ist die normale Reaktion des Editors, wenn der Schalter nicht explizit anders gesetzt wird.
  - Wenn der Wert des Schalters -1 ist, werden gefundene Zeilen automatisch protokolliert.
  - Wenn der Wert des Schalters n > 0 ist, so werden gefundene Zeilen automatisch protokolliert; dabei wird an der Stelle, an der der Zeiger dabei steht, jeweils das Zeichen eingeschoben, dessen Code-Wert im ZC1 gerade n ist.

Den Zustands-Schaltern wird ein Wert zugewiesen, indem dieser Wert als Parameter vor den Namen des Schalters geschrieben wird; z.B. erhaelt durch das Kommando "114ES" der Schalter "ES" den Wert 114.

1.2.7. Einfuegen von Texten

Wie schon im vorigen Abschnitt angegeben, werden Texte in den Editier-Puffer grundsaetzlich an der Stelle eingetragen, auf die der Puffer-Zeiger weist. Dabei wird der Zeiger nach jeder Eintragung um die Anzahl der eingetragenen Zeichen weitergeschoben, so dass aufeinanderfolgende Eintrage-Operationen den neuen Text jeweils an den gerade vorher eingetragenen anhaengen. Die verschiedenen Eintrage-Kommandos, die zur Verfuegung stehen, unterscheiden sich nur dadurch, dass sie es gestatten, Texte aus verschiedenen Quellen in den Puffer einzutragen:

- Das Kommando "I" traegt den Text des folgenden String-Parameters in den Puffer ein, wenn vor dem I kein numerischer Parameter steht.

- Das Kommando "^I" traegt analog den Text des folgenden String-Parameters ein; vor diesen Text wird jedoch ein Horizontal-Tabulator-Zeichen (HT, Control-I) eingefuegt.

- Das Kommando "nI$" traegt das Zeichen ein, dessen Wert im Code ZC1 gleich dem Wert des numerischen Parameters n ist.

- Das Kommando "nEA" traegt den Namen des n-ten Decks in der COSY-Datei in den Puffer ein.

- Das Kommando "n\" fuegt den Wert der Zahl n als Text in den Puffer ein.

- Das Kommando "^H" fuegt das aktuelle Datum in druckfertiger Form ein.

- Wenn im Q-Register i Text enthalten ist, so fuegt das Kommando "Gi" diesen Text in den Puffer ein. Dieses Kommando eignet sich daher besonders gut, mehrfach einzutragenden Text bequem an verschiedenen Stellen in den Puffer einzufuegen.

1.2.8. Aufbau von Strings

String-Parameter bestehen aus Folgen beliebiger Zeichen; sie duerfen lediglich ein bestimmtes Zeichen, das String-Begrenzungs-Zeichen, nicht enthalten. Ein String-Parameter beginnt direkt nach dem Kommando, zu dem er Parameter ist, bzw. direkt nach einem vorangehenden String-Begrenzungs-Zeichen. Der String endet entweder mit dem Ende der Eingabe an den Rechner oder, falls vorhanden, mit dem naechsten in dieser Eingabe stehenden String-Begrenzungs-Zeichen; dieses gehoert nicht mehr zum String selbst.

String-Begrenzungs-Zeichen ist normalerweise das Zeichen "$". Bei Einfuege- und Such-Kommandos kann es jedoch vorkommen, dass das Zeichen "$" selbst Bestandteil des Strings ist. In diesem Fall ist es notwendig, ein anderes Begrenzungs-Zeichen fuer den String zu waehlen. Dazu wird vor das betreffende Kommando der Modifikator "@" geschrieben, der veranlasst, dass das erste Zeichen hinter dem Kommando als String-Begrenzungs-Zeichen genommen wird. Zweckmaessigerweise waehlt man hierzu ein "exotisches" Zeichen, bei dem man einigermassen sicher ist, dass es im String selbst nicht auftritt.

Die Zeichen im String werden im allgemeinen genauso, wie sie angegeben werden, als Bestandteil des Strings uebernommen; speziell sind auch die in der Editor-Sprache bedeutungslosen Zeichen Zwischenraum (Blank) und Zeilenwechsel (Carriage return) Bestandteile des Strings und werden nicht uebersprungen. Ebenso werden in Strings kleine und grosse Buchstaben als verschiedene Zeichen betrachtet und entsprechend verschieden behandelt.

Da es noch viele Billig-Terminals gibt, die nicht ueber Gross-/Kleinschreibung verfuegen, sind im Editor Moeglichkeiten vorgesehen, bei Eintrage- und Such-Kommandos die zugehoerigen String-Parameter vor ihrer Verwendung durch das Kommando in verschiedener Weise zu transformieren, insbesondere auch durch geeignete Umschalt- und Ersatz-Zeichen die Eingabe von Zeichen, die in dem eingeschraenkten Zeichenvorrat des Terminals nicht vorhanden sind, zu spezifizieren. Dazu stehen im wesentlichen drei verschiedene Methoden zur Verfuegung:

- Setzen von Zustands-Schaltern, die bei allen folgenden Kommandos solche String-Transformationen durchfuehren; damit nicht mehr transformiert wird, muss der Schalter durch ein eigenes Kommando geloescht werden;
- Einfuegen von Steuerzeichen in den String, die alle folgenden Zeichen, soweit auf diese anwendbar, transfomieren; diese Steuerzeichen gelten bis zum Ende des betreffenden Strings, oder bis sie durch andere Steuerzeichen aufgehoben werden;
- Einfuegen von Steuerzeichen, die die Uebertragung des

naechsten Zeichens beeinflussen.

Der Zustands-Schalter "^V" transformiert in allen Strings in Eintrage- und Such-Kommandos Grossbuchstaben in Kleinbuchstaben, wenn er gesetzt ist, also den Wert 1 hat. Analog transformiert der Schalter "^W" Kleinbuchstaben in Grossbuchstaben, wenn er gesetzt ist. Setzen und Loeschen der Schalter geschieht dadurch, dass der neue Wert des Schalters als Parameter vor den Namen des Schalters gesetzt wird; z.B. wird der Schalter "^V" durch das Kommando "0^V" geloescht.

Das Steuerzeichen "^J" in einem String bewirkt, dass bis zum Ende des Strings alle folgenden Grossbuchstaben durch Kleinbuchstaben ersetzt werden; die Gueltigkeit dieses Steuerzeichens ist aber auf den String beschraenkt, in dem es auftritt. Analog ersetzt das Steuerzeichen "^K" alle nachfolgenden Kleinbuchstaben durch Grossbuchstaben. Jedes dieser beiden Steuerzeichen wird durch das andere aufgehoben; die Wirkung beider Zeichen kann durch das Steuerzeichen "^D" aufgehoben werden. Wenn zusaetzlich zu den Steuerzeichen noch Transformations-Zustands-Schalter gesetzt sind, so haben die Steuerzeichen die hoehere Prioritaet.

Das Steuerzeichen "^^" bewirkt, dass in dem betreffenden String alle folgenden Sonderzeichen @,[,\,],^ und _ durch die entsprechenden nicht-geshifteten ASCII-Zeichen `,{,|,},~ und DEL (Rub-out) ersetzt werden. Die Wirkung des Steuerzeichens "^^" wird durch ein zweites Steuerzeichen "^^" wieder aufgehoben.

Wenn nur ein Zeichen transformiert werden soll, ist es einfacher, diese Transformation durch das Steuerzeichen "^V" (von gross nach klein) bzw. "^W" (von klein nach gross) zu spezifizieren; diese Steuerzeichen, die innerhalb eines Strings auftreten, duerfen nicht mit den gleichnamigen Zustands-Schaltern, die durch eigene Kommandos gesetzt und geloescht werden, verwechselt werden. Von allen Transformationsregeln haben die Steuerzeichen "^V" und "^W" die hoechste Prioritaet.

Da es vorkommen kann, dass man Steuerzeichen selbst als Text in Strings eintragen will - dies wird vor allem dann der Fall sein, wenn Editor-Programme selbst editiert werden sollen -, gelten hierfuer eigene Regelungen:

- Eine Zeichen-Kombination "^x", wobei x ein beliebiger Grossbuchstabe ist, wird, wenn sie nicht eines der beschriebenen Steuerzeichen ist, in das entsprechende ASCII-Control-Character Control-x transformiert, z.B. "^D" in Control-D (EOT).
- Diese Transformation laesst sich auch fuer Steuerzeichen erreichen, wenn direkt vor das betreffende Steuerzeichen das Kontrollzeichen "^R" geschrieben wird. Das nachfolgende Steuerzeichen wird dann nicht als solches interpretiert, sondern in das zugehoerige ASCII-Control-Character umgewandelt und in den String eingetragen.

- Analog kann man alle im String folgenden Steuerzeichen (ausser "^R" und "^T") durch das Kontrollzeichen "^T" in ASCII-Control-Characters umwandeln lassen. Die Wirkung des Steuerzeichens "^T" wird durch ein zweites Steuerzeichen "^T" wieder aufgehoben.

Dem Zeichen "^" kommt somit innerhalb von Strings eine besondere Bedeutung zu, da dieses Zeichen in Verbindung mit dem nachfolgenden Zeichen eine ganze Reihe von Transformationen des Strings ermoeglicht. Diese Eigenschaft des Zeichens "^" erfordert es, das Zeichen selbst, wenn man es in einen String als Text und nicht zur Einleitung eines Steuerzeichens oder eines ASCII-Control-Characters eintragen will, durch eine spezielle Zeichen-Kombination zu ersetzen. Hierzu steht das Steuerzeichen "^Q" zur Verfuegung, das bewirkt, dass das nachfolgende Zeichen - auch wenn es das Zeichen "^" ist - unveraendert in den String uebernommen werden soll. Speziell ist somit das Zeichen "^" in Strings durch die Zeichenkombination "^Q^" darzustellen, wenn es als Text uebernommen werden soll.

Die hier genannten Moeglichkeiten, Strings zu tranformieren, werden im allgemeinen nur relativ selten benoetigt; sie werden jedoch fuer solche Faelle, in denen man gezwungen ist, an Terminals mit eingeschraenktem Zeichenvorrat zu arbeiten, zur Verfuegung gestellt, da ihre - wenn auch zugegebenermassen komplexe - Handhabung immer noch einfacher ist als die Eingabe der nicht vorhandenen Zeichen mittels Fluchtsymbol-Ersatzdarstellung. Ein Benutzer, der die Transformationen nicht benoetigt, kann davon ausgehen, dass Strings, die er eingibt, in genau der eingegebenen Form uebertragen werden, wenn er beruecksichtigt, dass er immer das Zeichen "^" durch "^Q^" ersetzen muss.

Strings, die Parameter in Such-Kommandos darstellen, koennen ausser den genannten Steuerzeichen noch weitere Steuerzeichen enthalten, die jedoch nicht zur Transformation des Suchstrings, sondern zur Spezifikation bestimmter Suchkriterien dienen. Diese Steuerzeichen stehen fuer Zeichen im zu suchenden String, die man nicht exakt angeben kann oder will:

- "^X" steht fuer jedes beliebige Zeichen
- "^Nx" steht fuer jedes beliebige Zeichen ausser dem Zeichen x, das dem "^N" unmittelbar folgt
- "^S" steht fuer jedes Trennzeichen (Separator), also fuer Zwischenraum (Blank), Komma, Semikolon, Carriage-return (CR), Line-Feed (LF), Horizontal-Tabulator (HT), Vertikal-Tabulator (VT) oder Form-Feed (FF)
- "^EA" steht fuer jeden Buchstaben
- "^EV" steht fuer jeden kleinen Buchstaben
- "^EW" steht fuer jeden grossen Buchstaben
- "^ED" steht fuer jede Ziffer
- "^EL" steht fuer jedes Zeilenwechsel-Zeichen (CR, LF, VT und FF)
- "^ES" steht fuer eine beliebige Folge von Blanks und HT

- "^E<nnn>" steht fuer das Zeichen, dessen Dezimalwert im ZC1 gleich nnn ist
- "^E<^Onnn>" steht fuer das Zeichen, dessen Oktalwert im ZC1 gleich nnn ist
- "^E<Qi>" steht fuer das Zeichen, dessen Wert im ZC1 als Zahlenwert im Q-Register i enthalten ist
- "^E[a,b,c,...]" steht fuer jedes beliebige der in der Klammer angegebenen Zeichen a,b,c,...

Durch einen weiteren Zustands-Schalter und ein weiteres Steuerzeichen laesst sich bei Suchstrings festlegen, ob zwischen Gross- und Kleinschreibung unterschieden werden soll oder nicht:

- Wenn der Zustands-Schalter "^X" gesetzt ist, also den Wert 1 hat, wird nach exakt dem angegebenen String gesucht; dies ist normalerweise der Fall. Wenn der Schalter jedoch geloescht, also durch das Kommando "0^X" auf den Wert 0 gesetzt wird, werden bei Such-Kommandos alle grossen Buchstaben als mit den entsprechenden kleinen Buchstaben identisch behandelt.
- Zusaetzlich kann durch das Steuerzeichen "^\" fuer alle in dem betreffenden String folgenden Buchstaben festgelegt werden, dass nicht zwischen Gross- und Kleinschreibung unterschieden werden soll. Die Wirkung des Steuerzeichens "^\" wird durch ein zweites Steuerzeichen "^\" wieder aufgehoben. Das Steuerzeichen "^\" hat eine hoehere Prioritaet als der Zustands-Schalter "^X".

1.2.9. Kontrollstrukturen

Mit den bis jetzt beschriebenen Kommandos ist es moeglich, Editier-Operationen zu spezifizieren und ein Editor-Programm als lineare Folge solcher Operationen zu schreiben. In vielen Faellen sind die durchzufuehrenden Editierungen jedoch nicht nur an einer Stelle des Textes durchzufuehren, sondern an mehreren Stellen, eventuell auch in Abhaengigkeit von irgendwelchen Kriterien. (Man denke etwa daran, dass in einem Programm, das ein bestimmtes Unterprogramm an mehreren Stellen aufruft, dieses Unterprogramm einen Parameter mehr oder weniger erhaelt und dass alle Aufrufe dieses Unterprogramms entsprechend geaendert werden muessen.) Um Editierungen dieser Art zu erleichtern, verfuegt der Editor ueber Kommandos, die es gestatten, den Kontroll-Fluss innerhalb des eingegebenen Editor-Programms zu beeinflussen.

Wenn ein Editor-Kommando oder eine Folge von Editor-Kommandos n-mal ausgefuehrt werden soll, so ist dieses Kommando bzw. diese Kommandofolge in spitze Klammern ("<" und ">") einzuschliessen; die Anzahl n, wie oft die Klammer ausgefuehrt werden soll, ist als numerischer Parameter vor die oeffnende Klammer zu schreiben. Diese Iterationsklammern duerfen geschachtelt sein; Klammerpaare muessen in einer Eingabe in jeder Ebene der Macro-Verschachtelung vollstaendig sein.

Wird eine Iterationsklammer ohne Angabe eines Wiederholungsfaktors spezifiziert, so bedeutet das Iteration ohne Begrenzung der Anzahl der Iterationsschritte. Da dies ohne weitere Massnahmen zu einer unendlichen Schleife im Editor-Programm fuehren wuerde, ist bei einer solchen unbeschraenkten Iteration die Angabe eines Abbruch-Kriteriums zwingend vorgeschrieben. Dabei sind zwei verschiedene Abbruch-Kriterien moeglich:

- Das Zeichen ";" fuehrt, wenn es mit einem numerischen Parameter versehen ist, zum Abbruch, wenn dieser Parameter 0 wird. Der Sprung erfolgt aus der aktuellen Iteration in die naechst aeussere bzw. in den linearen Teil des Editor-Programms, wenn es schon die aeusserste Iteration war.
- Wenn das Zeichen ";" keinen numerischen Parameter hat, muss es unmittelbar hinter einem Such-Kommando stehen; die Iteration wird dann verlassen, wenn der gesuchte String nicht gefunden wurde, die Suche also erfolglos war. Wenn der Erfolg einer Suche in dieser Weise zum Abbruch einer Iteration benuetzt wird, so erfolgt bei erfolgloser Suche keine Fehlermeldung, und das Editor-Programm wird auch nicht abgebrochen, sondern hinter der zugehoerigen schliessenden Iterationsklammer fortgesetzt.

Durch das Kommando "0" kann man die Fortfuehrung des Editor-Programms an der Stelle veranlassen, an der der String als Sprungmarke (Label) steht, der als String-Parameter im "0"-

Kommando angegeben ist. Das Label besteht aus der Zeichenfolge des Strings, ist aber zwischen Ausrufezeichen eingeschlossen. Da Labels sonst nicht im Editor-Programm ausgewertet werden und da auch nicht ueberprueft wird, ob durch ein "0"-Kommando auf sie gesprungen wird, stellt das Einschliessen beliebiger Zeichenfolgen in Ausrufezeichen eine Moeglichkeit zur Kommentierung von Editor-Programmen dar.

Das Ziel von Sprungbefehlen muss auf jeder Ebene der Macro-Verschachtelung ausserhalb aller Iterationen und bedingten Kommandos liegen; Spruenge auf ein Ziel in einer Iteration oder in einem bedingten Kommando sind auch von innerhalb dieser Iteration bzw. Bedingungsklammer nicht zulaessig! Sprungbefehle sollten daher nur verwendet werden, um aus allen Stufen einer Verschachtelung in den linearen Teil des Editor-Programms zurueckzukehren.

Man kann spezifizieren, dass ein Editor-Kommando bzw. eine Folge von Editor-Kommandos in Abhaengigkeit vom Wert eines numerischen Parameters ausgefuehrt werden soll. Dazu wird dieses Kommando bzw. diese Kommandofolge in sogenannte Bedingungsklammern eingeschlossen. Wenn die spezifizierte Bedingung erfuellt ist, wird der Inhalt der Bedingungsklammer ausgefuehrt, andernfalls nicht. Die oeffnende Bedingungsklammer besteht dabei aus dem numerischen Parameter, dessen Wert ueberprueft werden soll, gefolgt von einem Anfuehrungszeichen (") und einem Buchstaben, der die Werte des numerischen Parameters beschreibt, fuer die die Bedingung als erfuellt gelten soll. Die schliessende Bedingungsklammer besteht aus einem Apostroph ('). Auch Bedingungsklammer-Paare muessen in jeder Ebene der Macro-Verschachtelung vollstaendig sein. Die folgenden oeffnenden Bedingungsklammern stehen zur Verfuegung (angegeben ist immer, in welchem Fall der Inhalt der Klammer ausgefuehrt wird):

- n"E...' : n = 0
- n"N...' : n ≠ 0
- n"L...' : n < 0
- n"G...' : n > 0
- n"C...' : n ist der ZC1-Wert eines Zeichens, dem ein druckbares ASCII-Zeichen (ASCII- Wert > 31) entspricht
- n"D...' : n ist der ZC1-Wert einer Ziffer
- n"A...' : n ist der ZC1-Wert eines Buchstaben
- n"V...' : n ist der ZC1-Wert eines kleinen Buchstaben
- n"W...' : n ist der ZC1-Wert eines grossen Buchstaben
- n"T...' : n ist wahr (durch den Wert -1 repraesentiert)
- n"F...' : n ist falsch (durch den Wert 0 repraesentiert)
- n"S...' : n repraesentiert ein erfolgreiches Such-Kommando (das durch ":" modifiziert sein muss, damit es ein numerisches Ergebnis liefert)
- n"U...' : n repraesentiert eine erfolglose ":"-Suche

Der Inhalt der Bedingungsklammern wird jeweils dann ausgefuehrt, wenn die angegebene Bedingung erfuellt ist, andernfalls nicht.

1.2.10. Q-Register

Da es in Editor-Programmen und auch bei der ad-hoc-Eingabe haeufig vorkommt, dass man sich irgendwelche Zahlenwerte fuer die spaetere Verwendung in numerischen Parametern merken muss und da es auch zweckmaessig ist, sich gewisse Textstuecke separat vom Editier-Puffer aufheben zu koennen - etwa um sie an mehreren Stellen dort einfuegen zu koennen -, sieht die Editor-Sprache Speicherplaetze vor, die diese Informationen aufnehmen koennen. Diese Speicherplaetze, die als Q-Register bezeichnet werden, stellen in gewissem Sinn so etwas wie die Allzweck-Register in einem modernen Mehrregister-Rechner dar, doch ist die Laenge der Textstuecke, die ein Q-Register aufnehmen kann, nicht durch eine feste Schranke nach oben begrenzt, sondern nur durch die Menge des dem Editor insgesamt zur Verfuegung stehenden und noch nicht verbrauchten Speicherplatzes.

Ein Q-Register kann in drei verschiedenen Zustaenden sein:
- Es enthaelt eine Zahl.
- Es enthaelt ein Textstueck beliebiger Laenge.
- Es ist leer (aequivalent damit, dass es einen leeren String als Text enthaelt).

Wenn in ein Register eine Information eingeschrieben wird, so wird die vorher darin befindliche Information geloescht, und das Register wechselt gegebenenfalls seinen Zustand. So wird zum Beispiel eine Zahl aus einem Register geloescht, wenn man einen Text hineinschreibt, und umgekehrt; leeren kann man ein Register, indem man den leeren String als Text einschreibt.

Die Editor-Sprache sieht eine Reihe von Kommandos vor, um Q-Register zu manipulieren. In diesen Kommandos muss das zu bearbeitende Register durch seinen Namen spezifiziert werden, wenn es kein Register aus dem zusaetzlich vorhandenen Stack von 36 weiteren Q-Registern ist, auf die man aber nicht direkt zugreifen kann. An direkt zu bearbeitenden Registern stehen ebenfalls 36 zur Verfuegung, die durch die 10 Ziffern und die 26 Buchstaben bezeichnet werden; dabei wird nicht zwischen Gross- und Kleinbuchstaben unterschieden, so dass z.B. das Q-register "A" mit dem Q-Register "a" identisch ist. In der folgenden Beschreibung wird fuer den Register-Namen immer das Zeichen "i" angegeben; fuer dieses Zeichen kann also eine beliebige Ziffer oder ein beliebiger Buchstabe eingesetzt werden.

Fuer die Arbeit mit numerischen Werten in Registern stehen die folgenden Operationen zur Verfuegung:
- Durch das Kommando "nUi" wird der Wert des numerischen Parameters n in das Register i eingetragen. Er steht dort solange zur Verfuegung, bis das Register explizit durch ein Kommando ueberschrieben wird.

- Das Symbol "Qi" ist gleich dem numerischen Wert des Registers i, wenn dieses Register eine Zahl enthaelt; das Symbol kann in numerischen Parametern ueberall anstelle einer Zahl verwendet werden.
- Das Symbol "%i" entspricht dem Symbol "Qi", doch wird vor der Uebernahme des Wertes das Register i um 1 inkrementiert.

Man kann auf drei verschiedene Arten Texte in Q-Register einschreiben:
- Das Kommando "Vi" traegt den Text des folgenden String-Parameters in das Register i ein; ist der String-Parameter leer, so ist anschliessend auch das Register leer.
- Das Kommando "m,nXi" traegt alle Zeichen zwischen den Positionen m und n im Editier-Puffer in das Register i ein, also das m+1-te bis zum n-ten Zeichen einschliesslich; speziell traegt "HXi" den ganzen Puffer ein. Das Kommando "nXi" traegt dagegen alle Zeichen zwischen der aktuellen Position des Puffer-Zeigers und dem von dort gezaehlten n-ten Zeilenwechsel in das Register i ein; dabei werden bei positivem n die Zeilen vom Zeiger aus nach hinten gezaehlt, andernfalls nach vorn (wie bei den Befehlen "nL", "nK" und "nT"). Speziell wird durch den Befehl "0Xi" der Anfang der Zeile, in der der Zeiger steht, bis zum Zeiger in das Register i eingetragen; dagegen traegt der Befehl "Xi" oder "1Xi" alle Zeichen vom Zeiger bis zum Ende der Zeile, in der der Zeiger steht, ein, einschliesslich des die Zeile abschliessenden Zeilenwechsel-Zeichens.
- Schliesslich speichert das Kommando "*i" die letzte Eingabe an den Editor als Text in das Q-Register i. Dazu ist es jedoch erforderlich, dass die beiden Zeichen "*i" die beiden ersten Zeichen der neuen Eingabe sind; an allen anderen Stellen ist das Kommando "*i" illegal. Das Kommando im Register i kann dann spaeter bei Bedarf durch das Kommando "Mi" wieder zur Ausfuehrung gebracht werden, gegebenenfalls, nachdem es vorher, wie aller andere Text auch, geeignet editiert wurde. Man spart es sich auf diese Weise, mehrfach benoetigte Kommandofolgen jedesmal neu eingeben zu muessen, wenn man sie wieder benoetigt. Ebenso ist diese Dienstleistung ganz praktisch, wenn man eine laengere, aber falsche Eingabe nicht noch einmal schreiben will, da man sie dann verbessern und wiederholen kann, denn das Kommando "*i" kann auch dann ausgefuehrt werden, wenn die vorherige Eingabe zu einer Fehlermeldung gefuehrt hatte.

Texte in Q-Registern koennen auf zwei verschiedene Arten verwendet werden:
- Das Kommando "Gi" fuegt den Text an der Stelle, auf die der Puffer-Zeiger weist, in den Editier-Puffer ein. Dieses Kommando eignet sich dazu, mehrfach einzutragenden Text bequem an verschiedenen Stellen in den Puffer einzufuegen. Ausserdem laesst es sich zusammen mit den Kommandos "nXi" und "m,nXi" dazu verwenden, Textstuecke von einer Stelle nach einer anderen zu kopieren, auch unter Seitenwechsel oder von einem Deck in ein anderes.

- Das Kommando "Mi" fuehrt den im Q-Register i enthaltenen Text als Editor-Programm aus, ermoeglicht also die Verwendung von Macros in der Editor-Sprache. Innerhalb dieser Macros koennen andere Macros aufgerufen werden; ebenso duerfen auch Q-Register veraendert werden, mit Ausnahme derjenigen, die die Kommandos der aktuellen Macro-Verschachtelung enthalten.

Zusaetzlich zu den 36 direkt adressierbaren Q-Registern stehen weitere 36 Q-Register zur Verfuegung, die als Stack organisiert sind. Auf diese Register kann nur ueber eines der direkt adressierbaren Register zugegriffen werden, indem von diesem Daten in das oberste Stack-Register eingetragen werden (Push) oder indem die Daten aus dem obersten Stack-Register in das direkt adressierbare Register uebernommen werden (Pop); der Stack aendert dabei entsprechend seine Laenge. Jede Push-Operation "[i" fuegt dem Stack das neue oberste Register zu und schiebt die anderen Register um 1 nach unten, wobei die Laenge des Stacks um 1 waechst. Analog entnimmt jede Pop-Operation "]i" das oberste Register dem Stack und verkuerzt diesen dadurch um 1. Der Register-Inhalt wird dabei dem Register i entnommen bzw. in dieses eingetragen; dabei spielt es keine Rolle, ob dieser Inhalt aus einer Zahl oder einem String besteht. Der Register-Stack stellt fuer komplexere Editier-Vorgaenge einen ganz nuetzlichen Zusatz-Speicher in logischer Hinsicht dar, doch wird der verwendete Speicher natuerlich auch hier dem verbleibenden Freispeicher des Editors entnommen, der bei Verwendung des Register-Stacks nicht vergroessert wird.

1.2.11. Testhilfen

Wenn eine Eingabe an den Editor aus irgendeinem Grund fehlerhaft war, so wird ihre Abarbeitung an der Stelle abgebrochen, an der der Editor den Fehler erkennt. Der Editor gibt eine Fehlermeldung aus, die folgendermassen aufgebaut ist:

- Das erste Zeichen der Meldung besteht bzw. die beiden ersten Zeichen bestehen im allgemeinen aus dem fehlerhaften Editor-Kommando.
- Dann folgt der Text "Fehler:" sowie die interne Fehlernummer.
- Schliesslich folgt eine kurze verbale Beschreibung der Fehlerursache.

Bei Fehlern, die von der Datei-Schnittstelle des Editors erkannt wurden (z.B. wenn ein als Eingabe-File zu eroeffnendes COSY-Deck nicht existiert), wird in vielen Faellen vor der eigentlichen Fehlermeldung des Editors noch eine zusaetzliche Beschreibung des Fehlers von den Datei-Bearbeitungs-Routinen des Editors ausgegeben.

Gibt man als erstes Zeichen nach einer solchen Fehlermeldung das Zeichen "?" ein, so listet der Editor die vorherige Eingabe bis zu der Stelle auf, an der der Fehler erkannt wurde. Auf diese Weise kann man Fehler lokalisieren, deren Ursache aus der Fehlermeldung nicht klar erkennbar ist. Die Moeglichkeit, die vorherige Eingabe durch das Kommando "*i" in ein Q-Register i zu retten, geht dadurch nicht verloren; man muss dieses Kommando dann eben als erste Zeichen der naechsten Eingabe vorsehen.

Wenn das Kommando "?" an einer anderen Stelle eingegeben wird, so schaltet es einen Kommando-Trace im Editor ein, was ganz nuetzlich sein kann, wenn man sich - besonders bei komplizierteren Iterationen oder bedingten Kommandos - nicht im Klaren ist, welche Kommandos nun wirklich abgearbeitet werden, und in welcher Reihenfolge. Der Kommando-Trace kann durch ein zweites Kommando "?" wieder abgeschaltet werden; er endet jedoch spaetestens mit dem Ende der Eingabe.

Schliesslich sollte als weitere Testhilfe darauf hingewiesen werden, dass die genaue Kenntnis der verwendeten Editor-Kommandos viele Fehler von vorn herein vermeidbar macht. Deshalb ist es empfehlenswert, sich am Anfang auf eine geeignete Untermenge der Editor-Sprache zu beschraenken.

2. Die Editor-Sprache

2.1. Die Kommandos des Editors

2.1.1. File-Auswahl

ERfile$	'Edit Read'	Eingabe-File zum Lesen eroeffnen
EWfile$	'Edit Write'	Ausgabe-File erzeugen und zum Schreiben eroeffnen
EDfile$	'Edit Delete'	Eingabe-File zum Lesen eroeffnen und loeschen
EPfile$	'Edit Print'	Eingabe-File zum Lesen eroeffnen, allen Text auf die Texthaltungsdatei und/oder ins Protokoll ausgeben und den File wieder schliessen
EBfile$	'Edit Back-up'	Eingabe- und Ausgabe-File zur Bearbeitung eroeffnen mit Schutz durch Sicherheits-Kopie
EIfile$	'Edit Indirect'	File mit einer Kommandofolge zum Lesen eroeffnen, die Kommandofolge einlesen und ausfuehren. Schliesse den Kommando-File

2.1.2. Eingabe

Y	'Yank'	Puffer loeschen und eine Seite einlesen
A	'Append'	Seite einlesen und an vorhandenen Puffer anhaengen

2.1.3. Puffer-Positionen

B	'Before'	Vor dem ersten Zeichen; 0
.		Aktuelle Zeiger-Position; Anzahl der Zeichen links vom Puffer-Zeiger
Z		Ende des Puffers; Anzahl der Zeichen im Puffer
m,n		m+1-tes bis n-tes Zeichen im Puffer
H		Ganzer Puffer; B,Z

2.1.4. Argument-Operationen

m+n	Addition
m-n	Subtraktion
m*n	Multiplikation
m/n	Ganzzahlige Division
m&n	Logisches UND
m$n	Logisches ODER
( )	Fuehre eingeklammerte Operationen zuerst aus
^O	Die folgende Zahl ist oktal angegeben

2.1.5. Zeiger-Positionierung

nJ	'Jump'	Positioniere den Zeiger zwischen das n-te und das n+1-te Zeichen. J (ohne Parameter) entspricht 0J
nC	'Carry'	Positioniere den Zeiger um n Zeichen vorwaerts C entspricht 1C
nR	'Reverse'	Positioniere den Zeiger um n Zeichen rueckwaerts. R entspricht 1R, nR entspricht -nC
nL	'Line'	Positioniere den Zeiger auf den Anfang der n-ten Zeile von der aktuellen Zeiger-Position L entspricht 1L

2.1.6. Ausgabe

nT	'Type'	Gib allen Text von der aktuellen Zeiger-Position bis zum Anfang der n-ten Zeile von der aktuellen Zeiger-Position aus. T = 1T
m,nT	'Type'	Gib das m+1-te bis zum n-ten Zeichen aus
n=		Gib die Zahl n dezimal aus
n==		Gib die Zahl n oktal aus
1ET	'Edit Type'	Aendere den Ausgabe-Modus, so dass nicht-druckbare Zeichen nicht ersetzt werden
0ET	'Edit Type'	Aendere den Ausgabe-Modus, so dass nicht-druckbare Zeichen durch Ersatz-Darstellungen ausgegeben werden

0EU	'Edit Uppercase'	Gib kleine Buchstaben durch Ersatz-Darstellung aus
1EU	'Edit Uppercase'	Gib grosse Buchstaben durch Ersatz-Darstellung aus
-1EU	'Edit Uppercase'	Gib Buchstaben ohne Ersatz-Darstellung aus
-1ES	'Edit Search'	Gib nach String-Suche die gefundenen Zeilen aus
nES	'Edit Search'	Gib nach String-Suche die gefundenen Zeilen aus und markiere die Position des Zeigers durch das Zeichen mit dem Code-Wert n > 0
0ES	'Edit Search'	Gib nach String-Suche die gefundenen Zeilen nicht automatisch aus
0EA	'Edit All'	Gib eine Liste aller Files aus
^Atext^A		Gib den Text "text" aus
^Bi		Gib den Text im Q-Register i aus
^L		Gib ein Form-Feed aus

2.1.7. Loeschen

nD	'Delete'	Loesche n Zeichen hinter dem Zeiger. D = 1D
-nD	'Delete'	Loesche n Zeichen vor dem Zeiger
nK	'Kill'	Loesche alle Zeichen im Puffer vom Zeiger bis zum n-ten Zeilenwechsel vom Zeiger an. K = 1K
m,nK	'Kill'	Loesche das m+1-te bis zum n-ten Zeichen

2.1.8. Einfuegen

Itext$	'Insert'	Fuege den Text "text" zwischen I und $ am Zeiger ein und positioniere den Zeiger hinter die Einfuegung
^Itext$		Fuege den Text "text" mit einem einleitenden Tabulator-Zeichen (HT,^I) an der Zeiger-Position ein und positioniere den Zeiger hinter die Einfuegung
nI$	'Insert'	Fuege das Zeichen mit dem Code-Wert n ein

@I/text/	'Insert'	Fuege den durch das I unmittelbar folgende Zeichen begrenzten Text "text" am Zeiger ein. Das Kommando @ kann auch mit ^I verwendet werden
nEA	'Edit All'	Fuege den Namen des n-ten Files ein (n > 0)
n\		Fuege die Code-Darstellung der ganzen Zahl n ein
^H		Fuege das aktuelle Datum ein
1^V		Ersetze in allen folgenden Text-Argumenten Gross- durch Kleinschreibung. ^V entspricht 1^V
0^V		Schalte die Ersetzung durch Kleinschreibung ab
1^W		Ersetze in allen folgenden Text-Argumenten Klein- durch Grossschreibung. ^W entspricht 1^W
0^W		Schalte die Ersetzung durch Grossschreibung ab
^V		In Text-Argumenten: Ersetze das naechste Zeichen durch Kleinschreibung
^W		In Text-Argumenten: Ersetze das naechste Zeichen durch Grossschreibung
^J		Ersetze alle folgenden Zeichen durch Kleinschreibung. Wird durch ^D oder durch ^K aufgehoben
^K		Ersetze alle folgenden Zeichen durch Grossschreibung. Wird durch ^D oder durch ^J aufgehoben
^D		Uebernimm bei den folgenden Zeichen Gross- und Kleinschreibung wie angegeben
^^		In Text-Argumenten: Ersetze @,[,\,],^,_ durch die entsprechenden nicht-geshifteten Zeichen `,{,|,},~,DEL. Wird durch zweites ^^ aufgehoben
^R		Uebernimm das naechste Zeichen als Text. Dabei werden Zeichenkombinationen ^x als die entsprechenden Kontrollzeichen uebertragen
^T		In Text-Argumenten: Alle Kontroll-Zeichen ausser ^R und ^T sollen als Text uebernommen werden. Wird durch zweites ^T aufgehoben
^Q		Uebernimm das naechste Zeichen exakt, auch wenn es das Zeichen ^ ist

2.1.9. Ausgabe und Beendigung

PW	'Page Wait'	Gib den aktuellen Puffer aus
nP	'Page'	Gib den aktuellen Puffer aus, loesche den Puffer und lies den naechsten Puffer ein. Wiederhole diesen Vorgang, bis der n-te Puffer eingelesen ist. P entspricht 1P
m,nP	'Page'	Gib das m+1-te bis zum n-ten Zeichen aus. Aendere den Puffer nicht
EF	'End of File'	Schliesse den Ausgabe-File
^Z		Schliesse den Ausgabe-File und beende den Editor-Lauf
^C		Beende den Editor-Lauf
EX	'Exit'	Gib den Rest des Files aus und beende den Editor-Lauf
EL	'Exit List'	Gib den Rest des Files aus, gib den File in eine Texthaltungsdatei und/oder das Protokoll aus und beende den Editor-Lauf

2.1.10. Suche

nStext$	'Search'	Suche das n-te Auftreten des Textes "text" zwischen S und $ ab dem Zeiger, aber tausche keine Puffer aus. Stext$ entspricht 1Stext$
nFSt1$t2$		Suche das n-te Auftreten des Textes "t1" zwischen FS und $ ab dem Zeiger und ersetze ihn durch den Text "t2" zwsischen den beiden $; tausche keine Puffer aus. FS entspricht 1FS
nNtext$	'Nonstop'	Entspricht nStext$ ausser dass Puffer ausgetauscht werden, wenn der gesuchte Text im aktuellen Puffer nicht gefunden wird. N = 1N
nFNt1$t2$		Entspricht nFSt1$t2$ ausser dass Puffer ausgetauscht werden, wenn der gesuchte Text im aktuellen Puffer nicht gefunden wird. FN = 1FN
n_text$		Entspricht nNtext$ ausser dass Puffer nur eingelesen, aber nicht ausgeschrieben werden. _ entspricht 1_
n:Stext$		Entspricht nStext$ ausser dass bei erfolgreicher Suche ein Wert von -1 zurueckgegeben wird, andernfalls 0 anstatt einer Fehlermeldung. Das

	Kommando : kann auch mit FS, N, FN und _ verwendet werden
n@S/text/	Entspricht nStext$ ausser dass der Text von dem beliebigen Zeichen hinter S begrenzt wird. Das Kommando @ kann auch mit FS, N, FN und _ verwendet werden
0^X	Stelle den Such-Modus so ein, dass nicht zwischen Gross- und Kleinschreibung unterschieden wird
1^X	Stelle den Such-Modus auf exakte Entsprechung ein
1^V	Ersetze in allen folgenden Text-Argumenten Gross- durch Kleinschreibung. ^V entspricht 1^V
0^V	Schalte die Ersetzung durch Kleinschreibung ab
1^W	Ersetze in allen folgenden Text-Argumenten Klein- durch Grossschreibung. ^W entspricht 1^W
0^W	Schalte die Ersetzung durch Grossschreibung ab
^V	In Text-Argumenten: Ersetze das naechste Zeichen durch Kleinschreibung
^W	In Text-Argumenten: Ersetze das naechste Zeichen durch Grossschreibung
^J	Ersetze alle folgenden Zeichen durch Kleinschreibung. Wird durch ^D oder durch ^K aufgehoben
^K	Ersetze alle folgenden Zeichen durch Grossschreibung. Wird durch ^D oder durch ^J aufgehoben
^D	Uebernimm bei den folgenden Zeichen Gross- und Kleinschreibung wie angegeben
^^	In Text-Argumenten: Ersetze @,[,\,],^,_ durch die entsprechenden nicht-geshifteten Zeichen `,{,|,},~,DEL. Wird durch zweites ^^ aufgehoben
^R	Uebernimm das naechste Zeichen als Text. Dabei werden Zeichenkombinationen ^x als die entsprechenden Kontrollzeichen uebertragen
^T	In Text-Argumenten: Uebernimm alle Kontroll-Zeichen ausser ^R und ^T als Text. Wird durch zweites ^T aufgehoben

^\	In Text-Argumenten: Nimm bei allen folgenden Zeichen sowohl Gross- als auch Kleinschreibung an. Wird durch zweites ^\ aufgehoben
^X	In Text-Argumenten: Nimm auf dieser Position jedes Zeichen an
^S	Nimm auf dieser Position jedes Trennzeichen an
^Na	Nimm jedes Zeichen ausser dem beliebigen Zeichen "a" hinter ^N an
^Q	Uebernimm das naechste Zeichen exakt, auch wenn es das Zeichen ^ ist
^EA	Nimm jeden Buchstaben an
^EV	Nimm jeden kleinen Buchstaben an
^EW	Nimm jeden grossen Buchstaben an
^ED	Nimm jede Ziffer an
^EL	Nimm jedes Zeilenende-Zeichen an
^ES	Nimm jede Folge von Blanks und Tabs an
^E<nnn>	Nimm das Code-Zeichen mit dem Dezimal-Wert nnn an
^E<^Onnn>	Nimm das Code-Zeichen mit dem Oktal-Wert nnn an
^E<Qi>	Nimm das Code-Zeichen an, dessen Wert im Q-Register i steht
^E[a,b,c,...]	Nimm irgendeines der Zeichen in der Klammer an

2.1.11. Iteration und Flusskontrolle

n< >		Fuehre die eingeklammerten Kommandos n-mal aus. < > bedeutet Iteration ohne Beschraenkung der Anzahl der Iterationsschritte. Das Ende der Iteration muss durch ein Kommando ; spezifiziert werden
n;	'Break'	Falls n = 0, springe aus der aktuellen Iteration
;	'Break'	Springe aus der aktuellen Iteration, falls die zuletzt ausgefuehrte String-Suche nicht erfolgreich war

!label!	Definition einer Marke in der Kommandofolge oder Kommentar
0label$	Springe zu der durch !label! definierten Stelle
n"Ekommandos'	Falls n = 0, fuehre die zwischen "E und ' spezifierten Kommandos aus; anderenfalls springe zum '
n"Nkommandos'	Falls n NE 0, fuehre die eingeschlossenen Kommandos aus
n"Lkommandos'	Falls n < 0, fuehre die eingeschlossenen Kommandos aus
n"Gkommandos'	Falls n > 0, fuehre die eingeschlossenen Kommandos aus
n"Ckommandos'	Falls n der ASCII-Wert eines druckbaren Zeichens ist, fuehre die eingeschlossenen Kommandos aus
n"Dkommandos'	Falls n eine Ziffer ist, fuehre die eingeschlossenen Kommandos aus
n"Akommandos'	Falls n ein Buchstabe ist, fuehre die eingeschlossenen Kommandos aus
n"Vkommandos'	Falls n ein kleiner Buchstabe ist, fuehre die eingeschlossenen Kommandos aus
n"Wkommandos'	Falls n ein grosser Buchstabe ist, fuehre die eingeschlossenen Kommandos aus
n"Tkommandos'	Falls n wahr ('true') ist, fuehre die eingeschlossenen Kommandos aus
n"Fkommandos'	Falls n falsch ('false') ist, fuehre die eingeschlossenen Kommandos aus
n"Skommandos'	Falls n "erfolgreich" ist, fuehre die eingeschlossenen Kommandos aus
n"Ukommandos'	Falls n "nicht erfolgreich" ist, fuehre die eingeschlossenen Kommandos aus

2.1.12. Q-Register

nUi	'Use'	Speichere die ganze Zahl n in das Q-register i
Qi	'Q-register'	Gleich der im Q-Register i gespeicherten Zahl

%i		Inkrementiere den Wert im Q-Register i um 1 und gib diesen Wert zurueck
nXi	'Extract'	Speichere alle Zeichen von der aktuellen Zeiger-Position bis zum Anfang der n-ten Zeile von der aktuellen Zeiger-Position aus in das Q-Register i. Xi entspricht 1Xi
m,nXi	'Extract'	Speichere das m+1-te bis zum n-ten Zeichen in das Q-Register i
Vitext$		Speichere den Text "text" zwischen Vi und $ in das Q-Register i
@Vi/text/		Speichere den durch das Vi unmittelbar folgende Zeichen begrenzten Text "text" in das Q-Register i
Gi	'Get'	Fuege den Text im Q-Register i an der aktuellen Zeiger-Position ein und positioniere den Zeiger hinter die Einfuegung
Mi	'Macro'	Fuehre den Text im Q-Register i als Kommandofolge aus
[i	'Push'	Fuege den Inhalt des Q-Registers i in den Register-Stack ein
]i	'Pop'	Entferne den zuletzt in den Register-Stack eingefuegten Register-Inhalt aus dem Stack und uebertrage ihn in das Q-Register i
*i		Als erstes Kommando in einer Folge: Rette die vorhergehende Kommandofolge in das Q-Register i zur spaeteren Ausfuehrung als Makro

2.1.13. Spezielle numerische Werte

1A		Der (dezimale) Code-Wert des Zeichens hinter dem Zeiger
^E		Line-Feed Flag. Gleich 0 wenn bei der letzten Eingabe kein Line-Feed eingelesen wurde, sonst -1
^N		End-of-File Flag. Gleich -1 wenn bei der letzten Eingabe das Datei-Ende erreicht wurde, sonst 0
EA	'Edit All'	Gesamt-Anzahl der Files
EN	'Edit Number'	Nummer des Quell-Files

E0	'E0-Value'	Versionsnummer der Editor-Sprache (2 fuer die hier beschriebene Version, > 1000 fuer Implementierungen auf Fremdrechnern)
ET	'Edit Type'	Wert des Schalters ET fuer den Ausgabe-Modus
^X		Wert des Schalters ^X fuer den Suchmodus
EU	'Edit Uppercase'	Wert des Schalters EU fuer Ersatz-Darstellung von Buchstaben
^^x		Gleich dem (dezimalen) Code-Wert des beliebigen Zeichens "x" hinter dem ^^
\		Gleich dem Dezimal-Wert der Ziffernfolge hinter dem Zeiger
^T		Unterbrich die Abarbeitung der Kommandofolge, lies ein Zeichen ein und nimm den (dezimalen) Code-Wert dieses Zeichens an

2.1.14. Hilfen

?	Nach einer Fehlermeldung: Ausgabe der Kommandofolge bis einschliesslich zu dem Zeichen, da die Fehlermeldung verursacht hat
?	Schalte Trace-Modus ein. Ein zweites ? schaltet den Trace-Modus wieder aus

2.2. Fehlermeldungen

2.2.1. Fehlerstops

Wenn der Editor beim Start mit falschen Parametern versorgt wird - dies kann normalerweise nur dann vorkommen, wenn der Operator SB&EDIT direkt, also ohne Verwendung des Kommandos #EDIT, gestartet wird -, erfolgt sofortiger Programmabbruch mit Fehler. Das gleiche gilt, wenn versucht wird, eine Datei mit falscher Struktur oder eine andere als eine COSY-Datei zu bearbeiten. Der Editor beendet seinen Operatorlauf mit Fehler unter Ausgabe einer der folgenden Meldungen:

```
+++++ keine PHYS-Datei
+++++ Satzbau ist nicht G128W
+++++ Datei ist leer
+++++ keine Deck-Datei
+++++ Startsatzfehler
```

Von diesen Faellen abgesehen, sollte es nicht moeglich sein, einen Editor-Lauf anders als durch Eingabe eines der Ende-Kommandos "^C", "^Z", "EX" oder "EL" zu beenden. Sonstige Fehlersituationen, die waehrend eines Editor-Laufes auftreten, werden innerhalb dieses Laufes vom Editor behandelt, ohne den Operatorlauf deswegen zu beenden.

2.2.2. Fehlermeldungen des Editors

Wenn eine Eingabe an den Editor fehlerhaft war, so beendet der Editor die Abarbeitung dieser Eingabe an der Stelle, an der der Fehler erkannt wurde, und gibt eine Fehlermeldung aus. Diese Meldung enthaelt im wesentlichen die interne Fehlernummer dieses Fehlers sowie eine kurze verbale Beschreibung der Fehlerursache. Zusaetzlich wird am Anfang der Meldung das Kommando angegeben, das den Fehler hervorgerufen hat; diese Angabe ist jedoch nur dann von Bedeutung, wenn der Fehler durch ein existierendes Kommando verursacht wurde. Wenn die Eingabe ausserhalb von String-Parametern Zeichen enthaelt, die nicht zur Menge der erlaubten Kommandos gehoeren, so wird aus dem ersten dieser Zeichen eine Fehlernummer abgeleitet und daraus wieder das erste Zeichen der Fehlermeldung erzeugt; dies kann aufgrund verschiedener Umschluesselungen ein anderes Zeichen als das verbotenerweise eingegebene sein. Sieht man von diesem Sonderfall ab, so werden nur die in der folgenden Tabelle angegebenen Fehlermeldungen ausgegeben. Die Tabelle enthaelt jeweils das Kommando, das einen Fehler verursacht, sowie die zugehoerige Fehlernummer und den Text der Meldung. Bei Bedarf kann die Menge der Fehlermeldungen gegenueber dieser Tabelle erweitert werden, so dass dieser Abschnitt keinen Anspruch auf Vollstaendigkeit erheben kann. Die folgenden Fehlermeldungen stehen ueblicherweise zur Verfuegung:

^A	101	Ausgabe-Text fehlt
^B	201	illegaler oder fehlender Register-Name
^B	202	kein Text im Register
^C	301	File kann nicht geschlossen werden
^D	401	illegales Control-Kommando
^D	402	illegaler Control-Parameter
^H	801	Pufferueberlauf bei Datums-Speicherung
^I	901	Fehler bei einzutragendem String
^I	902	Pufferueberlauf bei Einfuegung
^0	1501	^0 ohne Oktal-Zahl
^Z	2601	File kann nicht geschlossen werden
^Z	2601	Ausgabe-File existiert nicht
^^	3001	unvollstaendiger ^^-Parameter
	3201	Kommando fehlt nach Argument
	3202	Kommando-Ende in Argument erreicht
	3203	illegales Kommando nach Doppel-Parameter
	3204	nicht alle Klammerpaare abgeschlossen
!	3301	Marke nicht abgeschlossen
"	3401	Sprung-Bedingung fehlt
"	3402	bedingtes Kommando fehlt
"	3403	Bedingungen zu tief geschachtelt
"	3404	Ende des bedingten Kommandos fehlt
"	3405	Argument der Bedingung fehlt
"	3406	illegale Bedingung
#	3501	illegales Kommando
$	3601	Operator an dieser Stelle illegal

$	3602	Ueberlauf des Operanden-Stacks
$	3603	illegaler unaerer Operator
%	3701	illegaler oder fehlender Register-Name
%	3702	Register enthaelt keine Zahl
&	3801	Operator an dieser Stelle illegal
&	3802	Ueberlauf des Operanden-Stacks
&	3803	illegaler unaerer Operator
'	3901	' ausserhalb eines bedingten Kommandos
(	4001	(an dieser Stelle illegal
(	4002	Ueberlauf des Operanden-Stacks
)	4101	) an dieser Stelle illegal
)	4102	Klammer-Struktur falsch
)	4103	illegaler Operator im Operanden-Stack
)	4104	arithmetischer Ueberlauf
*	4201	Operator an dieser Stelle illegal
*	4202	Ueberlauf des Operanden-Stacks
*	4203	illegaler unaerer Operator
*	4204	illegaler oder fehlender Register-Name
*	4205	Speichermangel bei Register
+	4301	Operator an dieser Stelle illegal
+	4302	Ueberlauf des Operanden-Stacks
,	4401	nicht alle Klammerpaare abgeschlossen
,	4402	mehr als zwei Parameter
-	4501	Operator an dieser Stelle illegal
-	4502	Ueberlauf des Operanden-Stacks
/	4701	Operator an dieser Stelle illegal
/	4702	Ueberlauf des Operanden-Stacks
/	4703	illegaler unaerer Operator
0	4801	Ueberlauf bei numerischem Parameter
:	5801	kein Such-Kommando nach :
;	5901	Ende der Iteration fehlt
;	5902	; ausserhalb einer Iteration
<	6001	unvollstaendige Iterations-Klammer
<	6002	Iterationen zu tief geschachtelt
<	6003	Ende der Iteration fehlt
<	6004	Kommando-Gruppe nicht abgeschlossen
=	6101	falsche Parameter-Anzahl bei =
>	6201	fehlerhafte Iteration
?	6301	ueberfluessiger Parameter bei ?
@	6401	kein String-Kommando nach @
A	6501	Eingabe-File erschoepft
A	6502	kein Eingabe-File vorhanden
A	6503	Zeiger steht am Pufferende
B	6601	Zahlenwert an falscher Stelle
B	6602	Ueberlauf des Operanden-Stacks
B	6603	Operanden-Stack leer
B	6604	illegaler Operator im Operanden-Stack
B	6605	arithmetischer Ueberlauf
C	6701	Zeigerpositionierung ausserhalb des Puffers
D	6801	Loeschversuch bis hinter den Puffer
D	6802	Loeschversuch bis vor den Puffer
E	6901	unvollstaendiges E-Kommando
E	6902	unvollstaendiger E-Parameter
F	7001	falsches oder unvollstaendiges F-Kommando
F	7002	Fehler bei Suchstring fuer F-Kommando

22. 9.78
We

F	7003	Ein- oder Ausgabe-File existiert nicht
F	7004	Fehler bei Ersatz-String
F	7005	Pufferueberlauf bei Ersetzung
F	7006	Suchstring nicht gefunden
G	7101	illegaler oder fehlender Register-Name
G	7102	kein Text im Register
G	7103	Pufferueberlauf bei Einkopieren
H	7201	kein Kommando nach H
H	7202	illegales Folge-Kommando nach H
I	7301	Fehler bei einzutragendem String
I	7302	Pufferueberlauf bei Einfuegung
I	7303	illegaler Code-Wert bei I
I	7304	I-Kommando nicht abgeschlossen
I	7305	Pufferueberlauf bei Zeichen-Speicherung
J	7401	Sprungziel ausserhalb des Puffers
M	7701	illegaler oder fehlender Register-Name
M	7702	kein Text im Register
M	7703	Makros zu tief verschachtelt
N	7801	Fehler bei Suchstring
N	7802	Suchstring kommt im Text nicht vor
N	7803	Ein- oder Ausgabe-File existiert nicht
O	7901	Fehler in Marke oder Kommentar
O	7902	Sprungziel existiert nicht
P	8001	Ausgabe-File existiert nicht
P	8002	Fehler bei Seitentransport
P	8003	Ausgabe-File existiert nicht
P	8004	Fehler bei Seitentransport
P	8005	Fehler bei Seitentransport
Q	8101	illegaler oder fehlender Register-Name
Q	8102	Register enthaelt keine Zahl
R	8201	Zeigerpositionierung ausserhalb des Puffers
S	8301	Fehler bei Suchstring
S	8302	Suchstring kommt im Text nicht vor
U	8501	illegaler oder fehlender Register-Name
U	8502	Speichern in Register illegal
U	8503	falsche Parameter-Anzahl bei Ui
V	8601	illegaler oder fehlender Register-Name
V	8602	Fehler bei String fuer Vi
V	8603	Speichern in Register nicht moeglich
V	8604	ueberfluessiger Parameter bei Vi
X	8801	kein Text im Puffer
X	8802	illegaler oder fehlender Register-Name
X	8803	Speichern in Register nicht moeglich
Y	8901	Eingabe-File erschoepft
Y	8902	kein Eingabe-File vorhanden
[	9101	illegaler oder fehlender Register-Name
[	9102	Speichermangel bei Register
[	9103	Register ist leer
[	9104	Register-Stack voll
[	9105	ueberfluessiger Parameter bei [
\	9201	Pufferueberlauf bei Zahlen-Speicherung
\	9202	keine oder zu grosse Zahl bei \-Parameter
]	9301	illegaler oder fehlender Register-Name
]	9302	Register-Stack leer
]	9303	Speichermangel bei Register

]	9304	ueberfluessiger Parameter bei]
^	9401	unvollstaendiges Control-Kommando
^	9402	unvollstaendiger Control-Parameter
_	9501	Fehler bei Suchstring
_	9502	Suchstring kommt im Text nicht vor
_	9503	Ein- oder Ausgabe-File existiert nicht
EA	9701	kein File mit dieser Nummer vorhanden
EA	9702	Pufferueberlauf bei Einfuegung
EB	9801	Fehler bei File-Name
EB	9802	Ausgabe-File existiert schon
EB	9803	Eingabe-File kann nicht eroeffnet werden
EB	9804	Ausgabe-File kann nicht erzeugt werden
EC	9901	illegales E-Kommando
EC	9902	illegaler E-Parameter
ED	10001	Fehler bei File-Name
ED	10002	File kann nicht geloescht werden
EF	10201	File kann nicht geschlossen werden
EF	10202	kein Ausgabe-File vorhanden
EI	10501	EI bei Back-up Editierung
EI	10502	EI bei indirekter Editierung
EI	10503	Fehler bei File-Name
EI	10504	Kommando-File kann nicht eroeffnet werden
EL	10801	Fehler bei Seitentransport
EL	10802	File kann nicht geschlossen werden
EL	10803	kein Ausgabe-File vorhanden
EL	10804	File existiert nicht in der Datei
EL	10805	Ausgabe nicht moeglich
EL	10806	Ausgabe abgebrochen
EP	11201	Fehler bei File-Name
EP	11202	File kann nicht eroeffnet werden
EP	11203	Ausgabe nicht moeglich
EP	11204	Ausgabe abgebrochen
ER	11401	Fehler bei File-Name
ER	11402	File kann nicht eroeffnet werden
ES	11501	illegaler Parameter-Wert bei ES
ES	11502	Parameter-Fehler bei ES
EW	11901	Fehler bei File-Name
EW	11902	Ausgabe-File kann nicht erzeugt werden
EW	11903	Ausgabe-File existiert schon
EX	12001	Fehler bei Seitentransport
EX	12002	File kann nicht geschlossen werden
EX	12003	kein Ausgabe-File vorhanden

2.2.3. Meldungen der Datei-Schnittstelle

Beim Start des Editors gibt die Datei-Schnittstelle bei Bedarf Meldungen aus, die den Benutzer informieren sollen, dass schon vor Anforderung der ersten Eingabe bestimmte Dienstleistungen erbracht wurden, und zwar:

- Erzeugen einer COSY-Struktur in einer leeren Datei;
- Erzeugen eines Decks, wenn der Parameter DECK im Kommando #EDIT besetzt war.

Die zugehoerigen Meldungen der Datei-Schnittstelle sind die folgenden:

```
***** Datei wurde initialisiert
***** Deck &&&&&&&&&&& wurde erzeugt
```

Ausserdem werden bei Fehlern, die von der Datei-Schnittstelle waehrend des Editor-Laufes erkannt werden, entsprechende Meldungen ausgegeben, sowie in bestimmten Faellen auch Warnungen zur Information des Benutzers. Die folgenden Texte koennen in diesem Zusammenhang ausgegeben werden:

```
+++++ Deck existiert nicht in der Datei
+++++ Deck wurde im Modus K6 erzeugt
+++++ Datei ist schreibgeschuetzt
+++++ zuviele Decks in der Datei
+++++ Deck existiert schon
+++++ Deck existiert nicht
+++++ Deck ist Ausgabe-File
+++++ Fehler im Leitblock
+++++ falsche Kartenart
+++++ falsche Blocknummer
+++++ falsche Checksumme
+++++ Puffer voll
+++++ Datei-Information nicht interpretierbar
+++++ kein Platz - Datei kann nicht erweitert werden
+++++ Warnung: Datei kann nicht ausreichend erweitert werden
+++++ Warnung: Datei ist schreibgeschuetzt
```

Einige dieser Meldungen fuehren zu einer anschliessenden Fehlermeldung des Editors mit Abbruch der Bearbeitung der Eingabe, andere werden nur zur Information ausgegeben und beeinflussen die weitere Bearbeitung nicht.

Bei einigen Operationen kann es vorkommen, dass der Editor nicht behebbare SSR-Fehler verursacht, etwa wenn versucht werden soll, in eine Texthaltungsdatei zu entzerren, auf die Schreibsperre gesetzt ist. In diesem Fall wird die betreffende SSR-Fehlermeldung ausgegeben und die Bearbeitung der Eingabe beendet; ein Abbruch des Editor-Laufes erfolgt nur dann, wenn dieser Fehler schon waehrend der Anfangsbehandlung, also vor der ersten Eingabe des Benutzers auftritt.

2.3. Hinweise fuer die Bedienung

2.3.1. Steuerung der Text-Files

Ueblicherweise wird das zu bearbeitende COSY-Deck durch geeignete Angabe des Parameters DECK im Kommando #EDIT ausgewaehlt. Wenn man einen neuen Text eintragen will, gibt man den Namen eines noch nicht existierenden Decks auf diesem Parameter an; der Editor erzeugt dann dieses Deck und erwartet die Eingabe von Text in das Deck. Wenn man existierenden Text veraendern will, gibt man den zugehoerigen Deck-Namen als Parameter DECK an; der Editor startet dann den im Abschnitt 1.2.2. beschriebenen Back-up-Mechanismus fuer dieses Deck.

In beiden Faellen kann dann das Eintragen neuen Textes und die Editierung alten Textes erfolgen. Diese Editierung kann innerhalb des aktuellen Editier-Puffers in beliebiger Reihenfolge an beliebigen Stellen erfolgen; hier gibt es keine Einschraenkungen in Bezug auf die Reihenfolge einzelner Editiervorgaenge, solange der verfuegbare Speicherplatz ausreicht. Wenn es jedoch notwendig ist, den Editierpuffer auszutauschen, so kann dies nur in Vorwaerts-Richtung geschehen; eine Rueckkehr zu einem Pufferinhalt, der schon in den Ausgabe-File ausgeschrieben wurde, ist nicht moeglich. Falls eine solche Rueckkehr zu einem frueheren Puffer einmal notwendig wird, muss der Editierlauf bis zum Ende der bearbeiteten Files durchgefuehrt werden - dies kann am einfachsten durch das Kommando "EX" erreicht werden -, und in einem weiteren Editierlauf muss dann wieder auf die gewuenschte Stelle positioniert werden. Wenn man die Beendigung des Operator-Laufes und den anschliessenden Wiederstart vermeiden will, so kann man denselben Effekt auch durch Ausgabe des Restes des Decks (etwa durch "100000P"), Abschliessen des Ausgabe-Files durch "EF" und anschliessende neue Back-up-Editierung durch "EB" erzielen.

Wenn man verschiedene Decks miteinander mischen will, ist es ratsam, auf die Back-up-Editierung zu verzichten und die zu bearbeitenden Decks explizit durch die Kommandos "ER" und "EW" zum Lesen bzw. Schreiben auszuwaehlen. Dabei ist zu beachten, dass Eingabe-Files ohne weitere Aktion durch Eroeffnen eines neuen Eingabe-Files gewechselt werden koennen, waehrend ein eroeffneter Ausgabe-File zuerst durch "EF" geschlossen werden muss, ehe ein neuer Ausgabe-File eroeffnet werden kann. Diese Regel gilt auch fuer Editierung im Back-up-Betrieb.

Wenn ein Back-up-Lauf aus irgendeinem Grund nicht ordnungsgemaess beendet wurde - d.h. wenn der Operatorlauf beendet wurde, ohne dass der Ausgabe-File durch eines der Kommandos "EF", "^Z", "EX" oder "EL" geschlossen wurde, stehen die bis

dahin erzeugten Ergebnisse in einem Deck mit dem (reservierten) Namen TEMP&BACK&UP. In diesem Fall kann - zur Sicherung dieser Daten - kein neuer Back-up-Lauf gestartet werden. Um nun den abgebrochenen Editier-Lauf fortzusetzen, geht man zweckmaessigerweise folgendermassen vor:

- Man erzeugt durch "EW" ein neues Deck, das die Ergebnisse des Laufes aufnehmen soll.

- Man eroeffnet das Deck TEMP&BACK&UP durch "ER" als Eingabe-File.

- Man liest mit einem Kommando "Y", gefolgt von einer geeigneten Anzahl von "P"-Kommandos, den Text des Eingabe-Files ein und schreibt ihn in den Ausgabe-File; dabei empfiehlt es sich, den so umkopierten Text sich wenigstens stueckweise durch "T"-Kommandos anzusehen, um festzustellen, wie weit dieser Text im Deck TEMP&BACK&UP enthalten ist.

- Man eroeffnet das Deck, das im abgebrochenen Back-up-Lauf bearbeitet wurde, durch "ER" als neuen Eingabe-File.

- Man sucht durch eine geeignete Folge von Such-Kommandos des Typs "_" das Ende des bis dahin schon ausgegebenen Textes, ohne dabei Ausgabe-Information zu produzieren.

- Dann wird durch das Kommando "0,.K" der Anfang des Editier-Puffers geloescht, da dieser ja schon ausgegebenen Text enthaelt.

- Ab jetzt kann der Editier-Vorgang wie ueblich fortgesetzt werden.

- Wenn man will, kann man abschliessend durch Umkopieren den Inhalt des neuen Ausgabe-Files in das urspruenglich bearbeitete Deck zurueckschreiben; dazu ist es jedoch notwendig, dieses Deck vorher zu loeschen. In jedem Fall kann das Deck TEMP&BACK&UP dann geloescht werden, um weitere Back-up-Laeufe wieder zu ermoeglichen.

2.3.2. Aufsuchen von Textstellen

Zum Aufsuchen einer bestimmten Stelle, an der ein Text bearbeitet werden soll, geht man am besten in mehreren Schritten vor:

- Zuerst sucht man eine markante Stelle im Text - z.B. die Deklaration der Routine, in der man etwas aendern will -, die vor der zu aendernden Stelle liegen sollte. Da bei laengeren Texten diese Suche im allgemeinen Puffer-Austausch beinhaltet, ist es empfehlenswert, die Suche durch ein "N"-Kommando zu starten.

- Nach dieser grossraeumigen Positionierung sucht man nun - am besten wieder mittels "N" - einen geeigneten String in der Naehe der zu veraendernden Stelle, zweckmaessigerweise wieder vor dieser, etwa eine Sprungmarke innerhalb der Routine. Dieser zweite Suchschritt kann oft auch entfallen.

- Von dort aus kann man dann durch ein "L"-Kommando auf den Anfang der zu veraendernden Zeile positionieren, falls man dies mit den vorherigen Schritten nicht schon getan hat.

- Innerhalb der Zeile kann man durch "C"- und "R"-Kommandos vor- und zurueckpositionieren; man kann aber auch durch Suche nach geeigneten Zeichengruppen den Zeiger auf eine beliebige Stelle der Zeile positionieren. Fuer diese Suche empfiehlt sich die Verwendung des Kommandos "S", damit man nicht durch Fehlbedienung - etwa einen Tipp-Fehler im Suchstring - einen ungewollten Pufferaustausch verursacht und damit die Positionierung wieder verliert.

Diese mehrstufige Form der Positionierung sieht recht umstaendlich und kompliziert aus, kann jedoch durch die Form der Editor-Sprache recht einfach realisiert werden, wie das folgende Beispiel zeigt:

Eine Text enthalte die Quellen mehrerer FORTRAN-Routinen. In der Routine DELETE soll 4 Zeilen hinter der Marke 1000 ein zusaetzlicher Summand in einen arithmetischen Ausdruck eingefuegt werden. Dazu ist es z.B. erforderlich, den Zeiger vor die dritte schliessende Klammer dieses Ausdrucks zu positionieren, da dort der neue Summand eingefuegt werden soll. Diese Positionierung kann durch die folgenden Kommandos erreicht werden:

```
NSUBROUTINE DELETE$   N 1000 $   4L   3S)$   R
```

Die Blanks zwischen den einzelnen Kommandos brauchen natuerlich nicht eingegeben zu werden; sie stoeren jedoch auch nicht.

Der hier gezeigte allgemeinste Fall der Positionierung vereinfacht sich in den meisten Faellen dadurch, dass entweder einer der Suchstrings schon so gewaehlt werden kann, dass man durch die Suche direkt auf die zu veraendernde Stelle positioniert, oder dass man bei mehreren aufeinanderfolgenden Editier-Operationen jeweils von der zuletzt bearbeiteten Stelle aus weiterpositioniert.

Wenn im obigen Beispiel etwa in der zu bearbeitenden Zeile ein Aufruf der Routine DLOG vorkommt, und wenn man sicher ist, dass dies im gesamten Text der zweite Aufruf dieser Routine ist, so kann man durch das Kommando

```
2NDLOG$
```

direkt auf diese Zeile positionieren. Ebenso kann man etwa direkt durch das Kommando "4L" auf diese Zeile positionieren, wenn man den Zeiger zuletzt in der Zeile mit der Marke 1000 stehen hatte.

Aus dem hier Gesagten geht hervor, dass die aktuelle Zeiger-Position ein wichtiger Parameter jedes Editier-Vorganges ist. Es ist daher oft noetig, die aktuelle Zeiger-Position fuer spaetere Verwendung festzuhalten. Da dieser Position ein numerischer Wert entspricht - naemlich die Anzahl der Zeichen im Puffer vor dem Zeiger -, kann man die aktuelle Position als numerischen Wert in ein Q-Register speichern, etwa durch das Kommando ".UN" in das Register N. Der Wert dieses Registers steht dann fuer spaetere Verwendung als numerischer Parameter in weiteren Editor-Kommandos als das Symbol "QN" zur Verfuegung. So kann man etwa zu einem spaeteren Zeitpunkt, falls bis dahin noch kein Puffer-Austausch vorgenommen wurde, durch das Kommando "QNJ" den Zeiger wieder auf die fruehere Position richten, oder man kann, wenn der Zeiger auf eine neue Position im Puffer weist, den Bereich zwischen der alten und der neuen Position durch das Kommando

```
QN,.T
```

bzw.

```
.,QNT
```

(jenachdem, ob die neue Position hinter oder vor der alten liegt) ausgeben oder durch

```
QN,.K
```

bzw.

```
.,QNK
```

loeschen.

Text wird grundsaetzlich an der Stelle in den Editier-Puffer eingetragen, auf die der Zeiger gerade weist. Beim Eintippen von Texten, bei denen man ein bestimmtes Druckbild durch Beginn der Zeilen in bestimmten Spalten erreichen will, ist es oft am einfachsten, den Zeiger vor dem Eintragen auf das Ende der vorhergehenden Zeile zu positionieren, da man dann als erstes Zeichen einen Zeilenwechsel eingeben muss und somit spaltenrichtig anfaengt. Auf das Zeilenende positio-

niert man, indem man durch "nL" auf den Anfang der naechsten Zeile positioniert und dann durch "R" vor den Zeilenwechsel zurueckgeht. Mehrere aufeinanderfolgende Eintragungen kann man auf diese Weise spaltenrichtig machen, indem man jede Eingabe ohne abschliessenden Zeilenwechsel abschickt und dann als erstes Zeichen im folgenden Eingabe-String (also im allgemeinen als erstes Zeichen nach dem folgenden "I"-Kommando) einen Zeilenwechsel eingibt. Da der Puffer-Zeiger nach jeder Eintragung hinter diese geschoben wird, fuehren aufeinanderfolgende Eingaben ohne weitere Positionierung zu einem fortlaufenden Text.

Zur Vermeidung von Missverstaendnissen sei hier noch einmal darauf hingewiesen, dass das Zeilenwechsel-Zeichen als letztes Zeichen einer Zeile aufgefasst wird, auch bei dem hier beschriebenen Verfahren des spaltenrichtigen Eintippens. Die Eingabe erfolgt in diesem Fall strenggenommen nicht zwischen zwei Zeilen, sondern die neuen Zeilen werden in die vorhergehende Zeile eingeschoben, so dass aus dieser einen Zeile nachher mehrere geworden sind. Dies ist hier ja ohne weiteres moeglich, da der Zeilenwechsel als normales Textzeichen aufgefasst wird und da somit Trennen und Zusammenfassen von Zeilen nur Einfuegen bzw. Loeschen von Zeilenwechsel-Zeichen ohne Umstrukturierung des Textes bedeutet.

Die beiden Schritte des Positionierens und des eigentlichen Editierens lassen sich fuer den Vorgang der Ersetzung eines Textstuecks durch ein anderees - gleicher oder vercshiedener Laenge - in vielen Faellen durch ein einziges Kommando "FS" bzw. "FN" durchfuehren. Dieses Kommando sucht den ersten der beiden als Parameter angegebenen Strings, loescht diesen String im Editier-Puffer und fuegt an seiner Stelle den als zweiten Parameter angegebenen String ein; der Zeiger steht anschliessend hinter dem so ersetzten String.

Wenn man etwa den fuenften Aufruf CALL DELETE durch den Aufruf CALL STORE ersetzen will, so kann dies durch das Kommando

```
5FSCALL DELETE$CALL STORE$
```

in einem Schritt geschehen. Im naechsten Abschnitt werden Methoden besprochen, mit denen man eine solche Ersetzung an allen Stellen im Text, wo der Suchstring vorkommt, durchfuehren kann.

2.3.3. Wiederholte Ausfuehrung von Kommandos

Es gibt im wesentlichen vier verschiedene Moeglichkeiten, Editier-Vorgaenge, die man mehrfach ausfuehren moechte, aber nur einmal spezifizieren will, durch geeignete Wiederholungskommandos mehrere Male anzustossen:

- Wenn man weiss, dass man eine Eingabe an den Editor spaeter in genau derselben Form noch einmal eingeben muss, so kann man diese Eingabe in eines der Q-Register fuer die spaetere Verwendung als Makro retten. Dazu gibt man als erstes Zeichen der naechsten Eingabe einen Stern ("*") ein, gefolgt von einem Register-Namen, etwa in der Form "*1". Die zuletzt eingegebene Kommando-Folge wird dann in das Q-Register, das man spezifiziert hat - in unserem Beispiel das Register 1 - eingetragen, ehe die neue Eingabe weiter abgearbeitet wird. Die dort gespeicherte Kommandofolge kann dann jederzeit durch einen Makro-Aufruf - im obigen Beispiel das Kommando "M1" - wieder zur Ausfuehrung gebracht werden.
  Diese Art der Wiederholungs-Anweisung ist auch dann nuetzlich, wenn man eine laengere Eingabe falsch gemacht hat. In diesem Fall kann man den Fehler beheben und die Eingabe durch den Makro-Aufruf wiederholen. Wenn man zum Beispiel auf die Zeile mit dem Suchstring "SEARCH" positionieren will und fuer diese Stelle eine komplexe Editierungs-Operation spezifiziert:

  ```
  SSEARCH$   ..............
  ```

  dabei aber den Zeiger schon hinter der gesuchten Stelle stehen hat, dann wird das Such-Kommando und mit ihm die folgende Editierung zurueckgewiesen, weil der Suchstring dann nicht mehr gefunden wird. Man kann den Schaden dann durch Eingabe der Kommando-Folge

  ```
  *1   J   M1
  ```

  beheben: Das alte Kommando wird durch "*1" in Register 1 gerettet; dann wird durch "J" der Zeiger auf den Puffer-Anfang gesetzt, und schliesslich wird das Kommando durch "M1" wieder zur Ausfuehrung gebracht.
  Aehnlich, wenn auch etwas umstaendlicher, kann man auch Kommandos reparieren, die in sich falsch sind:
  - Man rettet zuerst das falsche Kommando in ein Q-Register, etwa durch "*3" in Register 3.
  - Dann schreibt man den Inhalt des Editier-Puffers in ein anderes Q-Register, etwa durch "HXA" in Register A.
  - Hierauf loescht man den Editier-Puffer durch "HK".
  - In den nun leeren Puffer laedt man das falsche Kommando, in unserem Beispiel durch "G3". Dabei ist zu beachten, dass nach einem "G"-Kommando der Puffer-Zeiger immer hinter dem eingetragenen Text steht, in diesem Fall also

am Puffer-Ende. Es ist also zweckmaessig, fuer das weitere Vorgehen den Zeiger durch das Kommando "J" auf den Puffer-Anfang zu schieben.
- Das falsche Kommando kann nun, wie jeder andere Text, editiert und damit korrigiert werden.
- Anschliessend uebertraegt man das korrigierte Kommando zurueck in sein Register, im Beispiel durch "HX3" ins Register 3.
- Dann wird wieder durch "HK" der Puffer geloescht.
- Schliesslich wird der alte Puffer-Inhalt aus seinem Register in den Editier-Puffer gelesen, in unserem Fall durch "GA". Dabei ist wieder die Stellung des Puffer-Zeigers zu beachten.
- Wenn der Zeiger dann auf die richtige Stelle des Puffers weist, kann das fragliche Kommando durch Makro-Aufruf wiederholt werden, in unserem Beispiel durch "M1".

Es sei darauf hingewiesen, dass diese ganze Prozedur einige Uebung im Umgang mit dem Editor verlangt. Insbesondere benoetigt man einige Erfahrung, wenn waehrend dieser Korrektur-Phase noch einmal ein Fehler auftritt, da man dann bei einer falschen Reaktion leicht den Puffer-Inhalt zerstoert. Daher sei einem Neuling in der Arbeit mit dem Editor dringend abgeraten, solche Operationen durchzufuehren, wenn der Editier-Puffer wichtige, nicht restaurierbare Daten enthaelt.

- Eine zweite Moeglichkeit, Editier-Vorgaenge mehrfach ausfuehren zu lassen, ist dann gegeben, wenn man schon im Voraus weiss, dass eine bestimmte Operation n-mal ablaufen soll. Man spezifiziert dann fuer diese Operation n-fache Ausfuehrung, indem man sie in eine Iterationsklammer mit dem Wiederholungsfaktor n einschliesst. Wenn etwa an 10 Stellen hintereinander, an denen die Variable IBUF in einer Programm-Quelle auftritt, diese durch die Variable ABUFF zu ersetzen ist, so kann dies durch die Iterationsanweisung

```
10<FSIBUF$ABUFF$>
```

spezifiziert werden. Analog koennen auch Folgen von Editor-Kommandos durch Einschliessen in Iterationsklammern n-fach durchlaufen werden. Es ist dabei ohne weiteres moeglich, solche Iterationen zu schachteln, indem man die zugehoerigen Iterationsklammern schachtelt.

- In aehnlicher Weise kann man Editier-Vorgaenge, die an das Auftreten bestimmter Strings im Text gebunden sind, fuer jedes Auftreten dieser Strings im ganzen Text ausfuehren lassen. Man schliesst die zu iterierenden Kommandos dazu ebenfalls in Iterationsklammern ein, gibt aber keinen Wiederholungsfaktor vor der Klammer an. In diesem Fall hat man jedoch innerhalb der Iterationsklammer ein Such-Kommando, das auf den genannten String positioniert. Unmittelbar hinter dieses Such-Kommando fuegt man dann ein Semikolon (";") ein. Die Iteration wird dann solange wie-

derholt, bis die Suche ergebnislos wird; dann erfolgt allerdings keine Fehlermeldung, sondern nur ein Sprung aus dieser Iteration. Wenn man etwa alle die Zeilen im Editier-Puffer protokollieren will - dies geschieht durch Kommandos "OTT" -, die einen Unterprogramm-Aufruf in einem FORTRAN-Programm enthalten, so kann dies durch die Iterationsanweisung

```
<S CALL $; OTT>
```

spezifiziert werden. Analog kann man auch die Variable IBUF an allen Stellen im ganzen Text, an denen sie auftritt, durch die Variable ABUFF ersetzen, indem man die folgende Iteration spezifiziert (man beachte dabei den Unterschied zu dem Beispiel des vorigen Abschnitts!):

```
<FNIBUF$ABUFF$;>
```

Es ist auch bei Iterationen dieser Art ohne weiteres moeglich, geschachtelte Iterationen zu spezifizieren; ebenso kann man auch Iterationen beider Arten zusammen verwenden.

Analog kann man als Abbruchbedingung fuer eine solche unbeschraenkte Iteration auch spezifizieren, dass irgendein numerischer Parameter 0 werden soll. Dazu schreibt man hinter diesen Parameter ein Semikolon (";"); wie im vorigen Fall erfolgt dann ein Sprung aus der Iteration, wenn die Abbruch-Bedingung - in diesem Fall der Null-Durchgang des Parameters - erfuellt ist. Wenn man zum Beispiel die ZC1-Werte aller Zeichen im Puffer ausgeben lassen will, so kann dies durch die folgende Iterationsanweisung geschehen:

```
<.-Z; 1A= C>
```

Wenn der Zeiger am Puffer-Ende steht, ist der Wert von "." gleich der Anzahl aller Zeichen im Puffer, also gleich "Z", und somit ist der Ausdruck ".-Z" gleich 0, was zum Beenden der Iteration fuehrt. Andernfalls wird durch "1A=" der ZC1-Wert des naechsten Zeichens ausgegeben und durch "C" um ein Zeichen weiterpositioniert. Auch diese Form der Iteration kann geschachtelt und mit den beiden anderen Formen gemischt werden.

2.3.4. Kopieren von Text-Stuecken

Eine Operation, die man beim Editieren relativ haeufig benoetigt, ist das Verschieben einzelner Zeilen oder Bereiche von Zeilen. Hierzu lassen sich die Q-Register zweckmaessig einsetzen, und zwar sowohl als Transportmittel fuer die zu kopierenden Textteile als auch als Zwischenspeicher fuer die Grenzen dieser Bereiche. Die Bedienung der Register erfolgt am bequemsten etwas verschieden, in Abhaengigkeit davon, ob man eine bestimmte Anzahl von Zeilen kopieren will, oder ob man beliebig abgegrenzte Textbereiche bearbeiten will. Zunaechst soll hier die erste dieser Operationsweisen beschrieben werden:

- Wenn man n Zeilen im Text verschieben oder zusaetzlich an eine oder mehrere andere Stellen kopieren will, positioniert man zunaechst den Zeiger auf die im Abschnitt 2.3.2. beschriebene Weise auf den Anfang der ersten zu kopierenden Zeile. Das letzte Positionierungs-Kommando wird dabei im allgemeinen ein Kommando "mL" sein, das auf diesen Zeilenanfang positioniert.

- Dann kopiert man durch ein Kommando "nXi" die gewuenschten Zeilen in ein Q-Register. Wenn man zum Beispiel zwei Zeilen kopieren will, kann man diese durch "2XC" in das Register C uebertragen.

- Wenn die zu kopierenden Zeilen im Quell-Bereich geloescht werden sollen, so kann man dies unmittelbar anschliessend durch das Kommando "nK" tun, in unserem Beispiel durch das Kommando "2K".

- Dann positioniert man - oft wieder durch ein "mL"-Kommando - auf den Anfang der Zeile, vor die der zu kopierende Text eingeschoben werden soll. Man kann an dieser Stelle des Vorgangs auch Puffer austauschen und Ein- und/oder Ausgabe-File wechseln und auf diese Weise Textstuecke von einem Deck in ein anderes kopieren.

- Schliesslich kopiert man durch ein "Gi"-Kommando - im Beispiel durch das Kommando "GC" - den Text aus dem Register in den Editier-Puffer zurueck. Der Zeiger steht anschliessend direkt hinter dem einkopierten Text.

- Da das Register durch das "Gi"-Kommando nicht geloescht wird, kann man anschliessend bei Bedarf weiterpositionieren und den Text auch an andere Stellen eintragen.

Wenn man die beiden Zeilen des obigen Beispiels etwa um vier Zeilen weiterschieben und an der urspruenglichen Stelle loeschen will, so kann dies nach Positionierung auf den Anfang der ersten der beiden Zeilen durch die folgenden Kommandos geschehen (die Zwischenraeume zwischen den Kommandos sind

wieder nur der Uebersichtlichkeit wegen eingefuegt):

```
2XC   2K   4L   GC
```

Ein aehnliches Verfahren laesst sich anwenden, wenn man Textstuecke kopieren will, die keine ganze Anzahl von Zeilen umfassen. Man muss in diesem Fall lediglich Anfang und Ende der zu kopierenden Texte separat durch eigene Positionierungen bestimmen. Dazu geht man zweckmaessigerweise folgendermassen vor:

- Man positioniert zuerst auf den Anfang des zu kopierenden Bereiches und merkt sich diese Position in einem Q-Register, etwa durch das Kommando ".UN" im Register N.

- Dann positioniert man auf das Ende des zu kopierenden Bereiches.

- Jetzt kann man den Bereich zwischen der alten und der neuen Zeiger-Position durch ein Kommando des Typs "m,nXi" in ein weiteres Register eintragen, in unserem Beispiel durch das Kommando "QN,.XB" etwa in das Register B.

- Wenn der Quell-Bereich geloescht werden soll, so kann dies nun durch ein Kommando des Typs "m,nK" geschehen, im Beispiel durch das Kommando "QN,.K".

- Dann positioniert man auf die Stelle, an der der Bereich eingetragen werden soll. Auch hier kann man wieder Puffer austauschen oder die Files wechseln.

- Wie im anderen Fall kann jetzt der Bereich durch ein "Gi"-Kommando in den Text-Puffer eingetragen werden, im obigen Beispiel durch "GB". Der Zeiger steht anschliessend wieder hinter dem eingefuegten Text.

- Auch hier kann der kopierte Text durch weitere Positionierungen und weitere "Gi"-Kommandos mehrfach eingetragen werden.

Vergleicht man die beiden Formen der Kopier-Operation, so stellt man fest, dass sie sich nur in der Form der "X"- und "K"-Kommandos unterscheiden, wobei die zweite Form zwei Parameter benoetigt und daher im allgemeinen fuer einen dieser Werte die Verwendung eines weiteren Q-Registers erfordert.

Mit den hier besprochenen Operationen lassen sich im wesentlichen alle ueblicherweise benoetigten Editier-Vorgaenge durchfuehren. Speziell lassen sich auch alle vom Operator PS&TEXTHALT zur Verfuegung gestellten Operationen nachbilden, mit der Einschraenkung, dass eine Numerierung der Text-Zeilen hier nicht erfolgt, da sie einerseits ueberfluessig ist und da sie andererseits dem hier vertretenen Konzept der variablen Zeilengrenzen widerspricht.

3. Technische Realisierung des Editors

3.1. Zur Implementierung

3.1.1. Programm-Aufbau

Der Operator SB&EDIT ist in seiner derzeitigen Form (Version (4.08)) aus insgesamt 9 Montageobjekten aufgebaut. Vier dieser Montageobjekte bilden das eigentliche Editor-Programm:

- Das Montageobjekt TECO enthaelt das Hauptprogramm des Editors sowie eine Routine zum Einlesen der Editor-Kommandos. Das Hauptprogramm uebernimmt die Anfangsorganisation des Editor-Laufes, zu der die Umschluesselung und Ueberpruefung der im Kommando #EDIT uebergebenen Parameter gehoeren, sowie die Initialisierung der Status-Variablen des Editors. Ebenso wird hier die zu bearbeitende COSY-Datei eroeffnet, und die Alarmbehandlung wird initialisiert. Die weiteren Aktionen des Hauptprogramms bestehen dann lediglich aus dem abwechselnden Anstoss der Einlese-Routine TEXTIN (im gleichen Montageobjekt) und des Kommando-Interpreters ANALYSE (im Montageobjekt TEAN).
  Das Montageobjekt TECO ist teilweise in PS440 geschrieben, doch sind die Teile, die dem Betriebssystem naeher stehen, wie Startsatz-Interpretation und Datei-Behandlung, sowie aus Zeitgruenden auch die Umschluesselung in der Einlese-Routine durch TAS-Einschuebe realisiert.

- Das Montageobjekt TEAN besteht nur aus der Routine ANALYSE, die das im Eingabe-Puffer stehende Editor-Programm abarbeitet und auf Fehler ueberprueft. Jedesmal, wenn diese Routine eines der Editor-Kommandos erkannt hat und die zugehoerigen Parameter besorgt hat, fuehrt sie entweder das Kommando selbst aus oder stoesst die Ausfuehrung durch einen geeigneten Aufruf einer Routine aus einem der Montageobjekte TESUB oder TEFIL an.
  Aus Gruenden der Uebersichtlichkeit und der besseren Dokumentation - insbesondere im Hinblick auf die Uebertragung auf andere Rechner - ist das Montageobjekt TEAN vollstaendig in PS440 geschrieben.

- Das Montageobjekt TESUB enthaelt eine Reihe von Hilfsroutinen, die bei der Ausfuehrung der Editor-Kommandos von verschiedenen Stellen des Interpreters ANALYSE gebraucht werden:
  - ERROR : Ausgabe von Fehlermeldungen
  - COND : Ueberpruefung von numerischen Bedingungen
  - QNAME : Ueberpruefen und Bestimmen von Registernamen
  - QALOC : Platzreservierung fuer Q-Register
  - QFREI : Loeschen von Q-Registern
  - TEXTOUT : Textausgabe (auch mit Ersatz-Darstellungen)

- CONVERT : Umformen numerischer Werte in Textform
- SCRLF : Suchen von Zeilenwechseln
- CSKIP : Suchen nach abschliessenden Klammern
- STRING : Aufbereitung von String-Parametern
- SEARCH : Suchen von Strings im Eingabe-Text
- PARIN : Berechnung numerischer Parameterwerte

Auch das Montageobjekt TESUB ist aus den obengenannten Gruenden vollstaendig in PS440 geschrieben.

- Das Montageobjekt TEFIL stellt die Datei-Schnittstelle des Editors dar; es enthaelt alle Routinen, die zum Einlesen, Entzerren, Komprimieren und Ausschreiben der zu bearbeitenden Texte sowie zur Manipulation der COSY-Datei und der Decks in dieser Datei benoetigt werden:
  - PAGEIN : Einlesen von der COSY-Datei
  - PAGEOUT : Komprimieren und Ausschreiben auf die COSY-Datei sowie Schliessen des Ausgabe-Files
  - ROPEN : Eroeffnen des Eingabe-Files
  - WCREATE : Erzeugen des Ausgabe-Files
  - FNAME : Suchen eines Decks in der COSY-Datei
  - FLIST : Ausgabe von Deck-Namen
  - FDELETE : Loeschen von Decks
  - FCREATE : Eroeffnen und Erzeugen von Texthaltungsdateien
  - FCOPY : Entzerren in eine Texthaltungsdatei
  - FCLOSE : Bearbeitung einer Texthaltungsdatei beenden
  - XPAND : COSY-Darstellung entzerren
  - FSPACE : COSY-Datei bei Bedarf erweitern

  Die Routinen dieses Montageobjekts sind nur zum kleinen Teil in PS440 geschrieben; um Platz zu sparen und schnelle Abarbeitung zu erzielen, ist der groesste Teil durch TAS-Einschuebe realisiert.

Da die Schnittstelle eines PS440-Programms zum Betriebssystem weitgehend durch Routinen einer vorgegebenen Bibliothek, durch die sogenannten IOC-Dienste, realisiert wird, kommen zwei weitere Montageobjekte hinzu, die diese IOC-Dienste enthalten. Bei der Implementierung des Editors zeigte es sich allerdings, dass die standardmaessig zur Verfuegung stehenden IOC-Dienste sich durch hohen Platzbedarf, langsame Ausfuehrung und generelle Unzuverlaessigkeit auszeichnen. Aus diesem Grund wurden sie durch zwei Montageobjekte ersetzt, die genau die benoetigten Dienstleistungen in Form eigener TAS-Unterprogramme mit den Namen der zugehoerigen IOC-Dienste enthalten. Es handelt sich dabei um die beiden Montageobjekte:

- IOCOSB : Anfangs- und Ende-Organisation des Operatorlaufs
- IOC9SB : Ausgabe von Texten

Durch diese Aenderungen wurde der Operator um etwa 10 K kleiner und etwa um ein Drittel schneller.

Schliesslich gehoeren zum Operator noch die drei Montageobjekte

- S&ALARM : Behandlung von Alarmen
- S&SRF : Behandlung von SSR-Fehlern
- S&OPZEIT : Anfangs- und Ende-Meldung des Operators

aus der oeffentlichen Datenbasis.

Der Operator ist zur Zeit 20 K gross; davon entfallen 8 K auf Konstante und Befehle sowie 12 K auf Variable, von den letzteren wieder 11 K auf die verschiedenen Puffer des Editors, die in dieser Version des Operators noch statisch verwaltet werden.

3.1.2. Kapazitaet

Der Editor verfuegt ueber einen grossen Textpuffer, in dem der Editier-Puffer sowie die in Q-Registern gespeicherten Texte verwaltet werden, sowie ueber mehrere kleine Puffer, in denen Eingaben und String-Parameter gespeichert werden. Aus dem statisch festgelegten Umfang dieser Puffer ergeben sich bei der hier beschriebenen Version (4.08) des Editors bestimmte obere Grenzen fuer die Laengen der Texte, die auf einmal bearbeitet werden koennen:

- Im Editier-Puffer und in allen Q-Registern zusammen koennen nie mehr als 16 K Zeichen (16384 Zeichen) gespeichert werden; die Aufteilung auf den Editier-Puffer und die Q-Register ist dabei beliebig. Q-Register, die leer sind oder numerische Parameter enthalten, sind bei dieser Kapazitaets-Berechnung nicht zu beruecksichtigen, da sie keinen Platz im Textpuffer beanspruchen.

- Die Laenge einer Eingabe an den Editor ist auf eine Achtelseite (768 Zeichen) beschraenkt; dies gilt sowohl fuer Eingaben von der Konsole als auch Eingaben von Kommando-Files aus mittels des Kommandos "EI". Laengere Eingaben vom Terminal aus werden ignoriert, was im Saarbruecker ZVR-System insofern keine Einschraenkung darstellt, als hier Eingaben sowieso auf eine Achtelseite beschraenkt sind. Laengere Eingaben von einem Kommando-File aus werden nach 768 Zeichen abgeschnitten.

- String-Parameter werden im Editor in eine interne Darstellung ueberfuehrt, ehe sie in einem Kommando verwendet werden. Die Laenge dieser internen Darstellung ist auf 1 K (1024) Zeichen beschraenkt; diese Einschraenkung ist insofern keine echte Einschraenkung, da die String-Parameter ja aus irgendwelchen Eingaben an den Editor zu entnehmen sind, die ihrerseits schon auf 768 Zeichen beschraenkt sind.

Weitere Grenzwerte ergeben sich im Betrieb des Editors lediglich fuer die Groesse der bearbeiteten Dateien; diese Grenzwerte sind durch die jeweiligen Plattenspeicher- und/oder LFD-Berechtigungen des Benutzers gegeben.

In diesem Zusammenhang ist es noch von Interesse zu erwaehnen, dass der Plattenspeicher-Bedarf fuer die COSY-Datei

durch die Kompression des Textes im allgemeinen gegenueber dem fuer eine Texthaltungsdatei nach TR440-Konvention betraechtlich verringert ist. So liegen die Einsparungen z.B. bei einer typischen FORTRAN-Quelle bei etwa 40 %; bei strukturiert aufgebauten Programm-Quellen kann die Einsparung sogar bis zu 60 % gegenueber einer Texthaltungsdatei betragen. Bei extrem kurzen Texten naehert sich die Platzeinsparung auf der Platte dem fuer uebliche Anwendungen gueltigen Grenzwert von 94 %. Beruecksichtigt man noch zusaetzlich, dass eine einzige COSY-Deck-Datei bis zu 49 verschiedene Decks enthalten kann, von denen bei konventioneller Abspeicherung jedes einzelne eine eigene Datei waere, so sind die Einsparungen insgesamt sogar noch groesser. Dieser Aspekt ist besonders bei der LFD wichtig, da hier die Anzahl der in einem BKZ erlaubten Dateien ueblicherweise sehr begrenzt ist.

3.1.3. Dynamische Puffer-Verwaltung

Zur Zeit befindet sich eine Erweiterung des Editor-Programms in Arbeit, deren Ziel es ist, die im vorigen Abschnitt genannten Kapazitaets-Beschraenkungen aufzuheben. Ab der Version (5.00) wird der Editor mit einer dynamischen Pufferverwaltung versehen sein, die es gestattet, die einzelnen Puffer in ihrer Groesse den jeweiligen Erfordernissen anzupassen. Diese dynamische Pufferverwaltung hat zur Folge, dass die im letzten Abschnitt genannten absoluten Grenzwerte durch Werte ersetzt werden, die sich unter einer gewissen Grenze nur aus den Berechtigungen des Benutzers ergeben. Als absolute Grenze, die dann fuer alle vom Editor zu einer Zeit gespeicherten Texte zusammen gilt, sind 1 M (1048576) Zeichen vorgesehen. Diese Zahl bezieht sich selbstverstaendlich nur auf die Zeichen, die gerade vom Editor bearbeitet werden; die Anzahl der Zeichen in den bearbeiteten Dateien ist nach wie vor nur durch den verfuegbaren Platz auf dem Hintergrund begrenzt.

Unterhalb dieser absoluten Grenze werden die Puffer im Editor nach Bedarf dynamisch vergroessert und verkleinert. Werden die Puffer dabei insgesamt so gross, dass der Operator bei weiterer Vergroesserung die Kernspeicher-Berechtigung ueberschreiten wuerde, so wird eine weitere Vergroesserung des Puffers dadurch erreicht, dass der Puffer dann als virtueller Speicher gefuehrt wird. In diesem Fall befindet sich der Gesamt-Puffer, in Seiten unterteilt, auf dem Hintergrund, und nur die gerade zu bearbeitenden Seiten werden durch einen Paging-Algorithmus, der durch Page-Fault-Alarme gesteuert wird, in den Kernspeicher geladen. Die obere Grenze fuer die Groesse aller Puffer zusammen ergibt sich in diesem Fall auch wieder aus der Plattenspeicher-Berechtigung des Benutzers. Aus Effizienzgruenden sollte man von Puffern dieser Groesse jedoch nur im Notfall Gebrauch machen, da die Zeiten, die der Editor zum Suchen und zum Ver-

schieben von Textstuecken braucht, linear mit der Laenge des Editier-Puffers wachsen und bei virtuellem Pufferspeicher noch zusaetzlich um die Zeiten vergroessert werden, die fuer die Seitentransporte benoetigt werden.

Der Uebergang auf dynamische Pufferverwaltung hat ausser der Aufhebung der Kapazitaets-Grenzen noch den zusaetzlichen Effekt, dass der Operator SB&EDIT zwar um die Routinen zur dynamischen Speicherverwaltung vergroessert, gleichzeitig aber durch Wegfall der statisch vereinbarten Puffer im Variablen-Bereich erheblich verkleinert wird. Abschaetzungen haben ergeben, dass der Konstanten-Bereich des Editors in diesem Fall einen Umfang von etwa 8 oder 9 K haben wird, waehrend der Variablen-Bereich auf 1 K schrumpfen wird, so dass der Umfang des residenten Teils des Operators dann nur noch 9 oder 10 K betragen wird. Waehrend des Editor-Laufes kommt an Speicherbedarf zu diesem residenten Teil natuerlich noch der Bedarf fuer die Puffer hinzu; der Operator wird dann maximal so gross, dass er den zur Verfuegung stehenden Kernspeicher voll belegt.

22. 9.78
We

3.2. Erweiterung fuer off-line Betrieb

Es ist geplant, innerhalb des Saarbruecker ZVR-Systems die Funktionen der Texthaltung auf einen eigenen Rechner auszulagern, um so den Zentralrechner TR440 von Aufgaben zu entlasten, die weitgehend auch von kleineren Rechnern erledigt werden koennen. Unter Verwendung "intelligenter Arbeitsplaetze" (Terminals mit Mirkroprozessor und lokalem Speicher) kann dieses Prinzip der Auslagerung von Texthaltungsfunktionen auf kleinere Rechner noch weiter ausgedehnt werden, wenn man einen Teil der Editor-Operationen fuer off-line Verarbeitung direkt im Terminal zur Verfuegung stellt und gleichzeitig Operationen fuer den Transport von Textstuecken zwischen Terminal und Textverarbeitungs-Rechner vorsieht.

Der in einem solchen dezentralisierten Texthaltungssystem verwendete Editor muss nun einige spezielle Eigenschaften haben, um effizientes Arbeiten zu ermoeglichen. Speziell sollte der Transport von Textteilen zwischen Rechner und Peripherie moeglichst gering gehalten werden, was gleichzeitig zu der Forderung fuehrt, groessere Editierungsoperationen im off-line Modus ausfuehren zu koennen, ohne dazu zwischendurch dauernd Daten mit dem angeschlossenen Rechner austauschen zu muessen.

Der Editor SB&EDIT erfuellt diese Bedingungen, wenn er so erweitert wird, dass:
- einerseits die Bearbeitung des Editier-Puffers off-line im intelligenten Arbeitsplatz geschieht;
- andererseits die Datei-Schnittstelle und die Datenhaltung auf dem Texthaltungs-Rechner verbleibt;
- und schliesslich die Kommunikation zwischen diesen beiden Teilen durch Datenuebertragungsprogramme realisiert wird.

Dazu sind folgende Aenderungen und Erweiterungen des bestehenden Editor-Programms noetig:

- Der Kommando-Interpreter muss als Programm auf dem Prozessor des intelligenten Arbeitsplatzes laufen. Dazu ist es im wesentlichen erforderlich, die dazu gehoerenden Routinen (die Montageobjekte TEAN und TESUB sowie einen Teil des Montageobjekts TECO) auf diesem Prozessor zu programmieren.

- Statt der Routinen der Datei-Schnittstelle (im wesentlichen das Montageobjekt TEFIL) sind kurze Routinen fuer den Mikroprozessor zu schreiben, die ein Datenuebertragungsprogramm steuern, das dem Hauptrechner die folgenden Informationen uebermittelt:
  - die zu startende(n) Routine(n) der Datei-Schnittstelle auf diesem Hauptrechner mit geeigneter Parametrisierung;

22. 9.78
We

  - bei Bedarf Teile des lokalen Editier-Puffers;
  - die zu diesen Operationen noetigen Werte interner Zustandsvariablen des lokalen Editor-Programms.

- Auf dem zentralen Texthaltungs-Rechner laeuft dann die Datei-Schnittstelle des Editors wie im Fall der on-line Verarbeitung. Diese Datei-Schnittstelle wird jetzt aber nicht von einem Editor-Programm auf dem Zentral-Rechner bedient sondern von einem Datenuebertragungsprogramm, das dem auf dem intelligenten Arbeitsplatz entsprechen muss. Dieses Programm hat nun die Aufgabe, die vom Terminal empfangene Steuerinformation auszuwerten und die echte Datei-Schnittstelle im Zentral-Rechner geeignet zu bedienen sowie die verlangten Datentransporte (in beiden Richtungen) zu veranlassen.

Schliesslich kann die Datenhaltung voellig off-line geschehen, wenn der intelligente Arbeitsplatz zusaetzlich ueber eigene Peripherie (Floppy-Disk oder Magnetband-Kassette) verfuegt. In diesem Fall ist lediglich auch die Datei-Schnittstelle des Editors auf dem Mikroprozessor zu realisieren, was wohl keine allzugrosse Muehe machen duerfte, weil die verwendeten Datei-Strukturen extrem einfach sind - es werden zur Texthaltung ja nur physikalische Dateien verwendet, und auch diese werden rein sequentiell abgearbeitet. Der Editor sollte daher ohne grossen Aufwand auf jedes lokale System mit genuegendem Speicher uebertragbar sein, so dass ein universeller Einsatz dieses Konzepts auch in einem voellig heterogenen Rechner-Netz moeglich ist.

An der Universitaet Saarbruecken laufen zur Zeit Diplom-Arbeiten, die derartige Erweiterungen des hier beschriebenen Editors untersuchen und implementieren, zunaechst fuer das Mikroprozessor-System SBC 80/10 von INTEL auf der Basis des Prozessors INTEL 8080. Mit einem Abschluss dieser Arbeiten ist bis Anfang 1979 zu rechnen.

Abschliessend sei noch angemerkt, dass der Arbeitsaufwand zum Entwurf und zur Implementierung des hier beschriebenen Editor-Systems etwa ein halbes Mannjahr betrug; dabei wurden ausser einer Liste mit der Spezifikation der TECO-Kommandos (Anhang C der TECO-Beschreibung) keine weiteren Unterlagen benoetigt. Es ist daher zu erwarten, dass der Arbeitsaufwand zur Uebertragung des Editors auf einen beliebigen anderen Rechner bzw. auf ein Mikroprozessor-System bei etwa 2 bis 3 Mannmonaten liegt.

25. 9.78
We